MPTRAC

Generated by Doxygen 1.8.11

Contents

1 Main Page

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

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 atm_t Struct Reference

Atmospheric data.

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

Data Fields

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

4.1.1 Detailed Description

Atmospheric data.

Definition at line 459 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

Definition at line 462 of file libtrac.h.

4.1.2.2 double atm_t::time[NP]

Time [s].

Definition at line 465 of file libtrac.h.

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

Definition at line 468 of file libtrac.h.

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 471 of file libtrac.h.

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 474 of file libtrac.h.

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

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

Definition at line 477 of file libtrac.h.

4.1.2.7 double atm_t::up[NP]

Zonal wind perturbation [m/s].

Definition at line 480 of file libtrac.h.

4.1.2.8 double atm_t::vp[NP]

Meridional wind perturbation [m/s].

Definition at line 483 of file libtrac.h.

4.1.2.9 double atm_t::wp[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 486 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].
- 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).
- char csi_basename [LEN]

  Basename of CSI data files.
- double csi_dt_out

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

  Observation data file for CSI analysis.
- double csi_obsmin

  Minimum observation index to trigger detection.
- double csi_modmin

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

  Number of altitudes of gridded CSI data.
- double csi_z0

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

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

  Number of longitudes of gridded CSI data.
- double csi_lon0

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

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

  Number of latitudes of gridded CSI data.
- double csi_lat0

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

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

  Basename of grid data files.

- char grid_gpfile [LEN]

  Gnuplot file for gridded data.
- double grid_dt_out

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

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

  Number of altitudes of gridded data.
- double grid_z0

  Lower altitude of gridded data [km].
- double grid_z1

  Upper altitude of gridded data [km].
- int grid_nx

  Number of longitudes of gridded data.
- double grid_lon0

  Lower longitude of gridded data [deg].
- double grid_lon1

  Upper longitude of gridded data [deg].
- int grid_ny

  Number of latitudes of gridded data.
- double grid_lat0

  Lower latitude of gridded data [deg].
- double grid_lat1

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

  Basename for profile output file.
- char prof_obsfile [LEN]

  Observation data file for profile output.
- int prof_nz

  Number of altitudes of gridded profile data.
- double prof_z0

  Lower altitude of gridded profile data [km].
- double prof_z1

  Upper altitude of gridded profile data [km].
- int prof_nx

  Number of longitudes of gridded profile data.
- double prof_lon0

  Lower longitude of gridded profile data [deg].
- double prof_lon1

  Upper longitude of gridded profile data [deg].
- int prof_ny

  Number of latitudes of gridded profile data.
- double prof_lat0

  Lower latitude of gridded profile data [deg].
- double prof_lat1

  Upper latitude of gridded profile data [deg].
- char ens_basename [LEN]

  Basename of ensemble data file.
- char stat_basename [LEN]

  Basename of station data file.
- double stat_lon

Longitude of station [deg].

- double stat_lat

  Latitude of station [deg].

- double stat_r

  Search radius around station [km].

4.2.1 Detailed Description

Control parameters.

Definition at line 177 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 180 of file libtrac.h.

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

Quantity names.

Definition at line 183 of file libtrac.h.

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

Quantity units.

Definition at line 186 of file libtrac.h.

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

Quantity output format.

Definition at line 189 of file libtrac.h.

4.2.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 192 of file libtrac.h.

4.2.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 195 of file libtrac.h.

4.2.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 198 of file libtrac.h.

4.2.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 201 of file libtrac.h.

4.2.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 204 of file libtrac.h.

4.2.2.10 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 207 of file libtrac.h.

4.2.2.11 int ctl_t::qnt_t

Quantity array index for temperature.

Definition at line 210 of file libtrac.h.

4.2.2.12 int ctl_t::qnt_u

Quantity array index for zonal wind.

Definition at line 213 of file libtrac.h.

4.2.2.13 int ctl_t::qnt_v

Quantity array index for meridional wind.

Definition at line 216 of file libtrac.h.

4.2.2.14 int ctl_t::qnt_w

Quantity array index for vertical velocity.

Definition at line 219 of file libtrac.h.

4.2.2.15 int ctl_t::qnt_h2o

Quantity array index for water vapor vmr.

Definition at line 222 of file libtrac.h.

4.2.2.16 int ctl_t::qnt_o3

Quantity array index for ozone vmr.

Definition at line 225 of file libtrac.h.

4.2.2.17 int ctl_t::qnt_theta

Quantity array index for potential temperature.

Definition at line 228 of file libtrac.h.

4.2.2.18 int ctl_t::qnt_pv

Quantity array index for potential vorticity.

Definition at line 231 of file libtrac.h.

4.2.2.19 int ctl_t::qnt_tice

Quantity array index for T_ice.

Definition at line 234 of file libtrac.h.

4.2.2.20 int ctl_t::qnt_tsts

Quantity array index for T_STS.

Definition at line 237 of file libtrac.h.

4.2.2.21 int ctl_t::qnt_tnat

Quantity array index for T_NAT.

Definition at line 240 of file libtrac.h.

4.2.2.22 int ctl_t::qnt_stat

Quantity array index for station flag.

Definition at line 243 of file libtrac.h.

4.2.2.23 int ctl_t::qnt_gw_u750

Quantity array index for low-level zonal wind.

Definition at line 246 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 249 of file libtrac.h.

4.2.2.25 int ctl_t::qnt_gw_sso

Quantity array index for subgrid-scale orography.

Definition at line 252 of file libtrac.h.

4.2.2.26 int ctl_t::qnt_gw_var

Quantity array index for gravity wave variances.

Definition at line 255 of file libtrac.h.

4.2.2.27 int ctl_t::direction

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

Definition at line 258 of file libtrac.h.

4.2.2.28 double ctl_t::t_start

Start time of simulation [s].

Definition at line 261 of file libtrac.h.

4.2.2.29 double ctl_t::t_stop

Stop time of simulation [s].

Definition at line 264 of file libtrac.h.

4.2.2.30 double ctl_t::dt_mod

Time step of simulation [s].

Definition at line 267 of file libtrac.h.

4.2.2.31 double ctl_t::dt_met

Time step of meteorological data [s].

Definition at line 270 of file libtrac.h.

4.2.2.32 int ctl_t::met_np

Number of target pressure levels.

Definition at line 273 of file libtrac.h.

4.2.2.33 double ctl_t::met_p[EP]

Target pressure levels [hPa].

Definition at line 276 of file libtrac.h.

4.2.2.34 int ctl_t::isosurf

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

Definition at line 280 of file libtrac.h.

4.2.2.35 char ctl_t::balloon[LEN]

Balloon position filename.

Definition at line 283 of file libtrac.h.

4.2.2.36 double ctl_t::turb_dx_trop

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

Definition at line 286 of file libtrac.h.

4.2.2.37 double ctl_t::turb_dx_strat

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

Definition at line 289 of file libtrac.h.

4.2.2.38 double ctl_t::turb_dz_trop

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

Definition at line 292 of file libtrac.h.

4.2.2.39 double ctl_t::turb_dz_strat

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

Definition at line 295 of file libtrac.h.

4.2.2.40 double ctl_t::turb_meso

Scaling factor for mesoscale wind fluctuations.

Definition at line 298 of file libtrac.h.

4.2.2.41 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 301 of file libtrac.h.

4.2.2.42 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 304 of file libtrac.h.

4.2.2.43 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 307 of file libtrac.h.

4.2.2.44 double ctl_t::psc_hno3

HNO3 volume mixing ratio for PSC analysis.

Definition at line 310 of file libtrac.h.

4.2.2.45 char ctl_t::gw_basename[LEN]

Basename for gravity wave variance data.

Definition at line 313 of file libtrac.h.

4.2.2.46 char ctl_t::atm_basename[LEN]

Basename of atmospheric data files.

Definition at line 316 of file libtrac.h.

4.2.2.47 char ctl_t::atm_gpfile[LEN]

Gnuplot file for atmospheric data.

Definition at line 319 of file libtrac.h.

4.2.2.48 double ctl_t::atm_dt_out

Time step for atmospheric data output [s].

Definition at line 322 of file libtrac.h.

4.2.2.49 int ctl_t::atm_filter

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

Definition at line 325 of file libtrac.h.

4.2.2.50 char ctl_t::csi_basename[LEN]

Basename of CSI data files.

Definition at line 328 of file libtrac.h.

4.2.2.51 double ctl_t::csi_dt_out

Time step for CSI data output [s].

Definition at line 331 of file libtrac.h.

4.2.2.52 char ctl_t::csi_obsfile[LEN]

Observation data file for CSI analysis.

Definition at line 334 of file libtrac.h.

4.2.2.53 double ctl_t::csi_obsmin

Minimum observation index to trigger detection.

Definition at line 337 of file libtrac.h.

4.2.2.54 double ctl_t::csi_modmin

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

Definition at line 340 of file libtrac.h.

4.2.2.55 int ctl_t::csi_nz

Number of altitudes of gridded CSI data.

Definition at line 343 of file libtrac.h.

4.2.2.56 double ctl_t::csi_z0

Lower altitude of gridded CSI data [km].

Definition at line 346 of file libtrac.h.

4.2.2.57 double ctl_t::csi_z1

Upper altitude of gridded CSI data [km].

Definition at line 349 of file libtrac.h.

4.2.2.58 int ctl_t::csi_nx

Number of longitudes of gridded CSI data.

Definition at line 352 of file libtrac.h.

4.2.2.59 double ctl_t::csi_lon0

Lower longitude of gridded CSI data [deg].

Definition at line 355 of file libtrac.h.

4.2.2.60 double ctl_t::csi_lon1

Upper longitude of gridded CSI data [deg].

Definition at line 358 of file libtrac.h.

4.2.2.61 int ctl_t::csi_ny

Number of latitudes of gridded CSI data.

Definition at line 361 of file libtrac.h.

4.2.2.62 double ctl_t::csi_lat0

Lower latitude of gridded CSI data [deg].

Definition at line 364 of file libtrac.h.

4.2.2.63 double ctl_t::csi_lat1

Upper latitude of gridded CSI data [deg].

Definition at line 367 of file libtrac.h.

4.2.2.64 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 370 of file libtrac.h.

4.2.2.65 char ctl_t::grid_gpfile[LEN]

Gnuplot file for gridded data.

Definition at line 373 of file libtrac.h.

4.2.2.66 double ctl_t::grid_dt_out

Time step for gridded data output [s].

Definition at line 376 of file libtrac.h.

4.2.2.67 int ctl_t::grid_sparse

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

Definition at line 379 of file libtrac.h.

4.2.2.68 int ctl_t::grid_nz

Number of altitudes of gridded data.

Definition at line 382 of file libtrac.h.

4.2.2.69 double ctl_t::grid_z0

Lower altitude of gridded data [km].

Definition at line 385 of file libtrac.h.

4.2.2.70 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 388 of file libtrac.h.

4.2.2.71 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 391 of file libtrac.h.

4.2.2.72 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 394 of file libtrac.h.

4.2.2.73 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 397 of file libtrac.h.

4.2.2.74 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 400 of file libtrac.h.

4.2.2.75 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 403 of file libtrac.h.

4.2.2.76 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 406 of file libtrac.h.

4.2.2.77 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 409 of file libtrac.h.

4.2.2.78 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 412 of file libtrac.h.

4.2.2.79 int ctl_t::prof_nz

Number of altitudes of gridded profile data.

Definition at line 415 of file libtrac.h.

4.2.2.80 double ctl_t::prof_z0

Lower altitude of gridded profile data [km].

Definition at line 418 of file libtrac.h.

4.2.2.81 double ctl_t::prof_z1

Upper altitude of gridded profile data [km].

Definition at line 421 of file libtrac.h.

4.2.2.82 int ctl_t::prof_nx

Number of longitudes of gridded profile data.

Definition at line 424 of file libtrac.h.

4.2.2.83 double ctl_t::prof_lon0

Lower longitude of gridded profile data [deg].

Definition at line 427 of file libtrac.h.

4.2.2.84 double ctl_t::prof_lon1

Upper longitude of gridded profile data [deg].

Definition at line 430 of file libtrac.h.

4.2.2.85 int ctl_t::prof_ny

Number of latitudes of gridded profile data.

Definition at line 433 of file libtrac.h.

4.2.2.86 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 436 of file libtrac.h.

4.2.2.87 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 439 of file libtrac.h.

4.2.2.88 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 442 of file libtrac.h.

4.2.2.89 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 445 of file libtrac.h.

4.2.2.90 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 448 of file libtrac.h.

4.2.2.91 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 451 of file libtrac.h.

4.2.2.92 double ctl_t::stat_r

Search radius around station [km].

Definition at line 454 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 491 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 double met_t::time

Time [s].

Definition at line 494 of file libtrac.h.

4.3.2.2 int met_t::nx

Number of longitudes.

Definition at line 497 of file libtrac.h.

4.3.2.3 int met_t::ny

Number of latitudes.

Definition at line 500 of file libtrac.h.

4.3.2.4 int met_t::np

Number of pressure levels.

Definition at line 503 of file libtrac.h.

4.3.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 506 of file libtrac.h.

4.3.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 509 of file libtrac.h.

4.3.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 512 of file libtrac.h.

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

Surface pressure [hPa].

Definition at line 515 of file libtrac.h.

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

Pressure on model levels [hPa].

Definition at line 518 of file libtrac.h.

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

Temperature [K].

Definition at line 521 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 524 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 527 of file libtrac.h.

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

Vertical wind [hPa/s].

Definition at line 530 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 533 of file libtrac.h.

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

Ozone volume mixing ratio [1].

Definition at line 536 of file libtrac.h.

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

- libtrac.h

5 File Documentation

5.1 center.c File Reference

Calculate center of mass of air parcels.

Functions

- int main (int argc, char ∗argv[ ])

5.1.1 Detailed Description

Calculate center of mass of air parcels.

Definition in file center.c.

5.1.2 Function Documentation

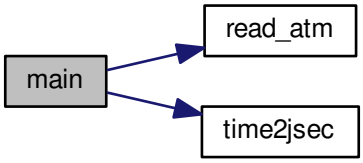
5.1.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 28 of file center.c.

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

```
00106       latm += atm->lat[ip] / atm->np;
00107       zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00108       lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00109       lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00110     }
00111
00112     /* Normalize... */
00113     zs = sqrt(zs - gsl_pow_2(zm));
00114     lons = sqrt(lons - gsl_pow_2(lonm));
00115     lats = sqrt(lats - gsl_pow_2(latm));
00116
00117     /* Sort arrays... */
00118     gsl_sort(atm->p, 1, (size_t) atm->np);
00119     gsl_sort(atm->lon, 1, (size_t) atm->np);
00120     gsl_sort(atm->lat, 1, (size_t) atm->np);
00121
00122     /* Get time from filename... */
00123     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00124     year = atoi(tstr);
00125     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00126     mon = atoi(tstr);
00127     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00128     day = atoi(tstr);
00129     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00130     hour = atoi(tstr);
00131     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00132     min = atoi(tstr);
00133     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00134
00135     /* Write data... */
00136     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g "
00137             "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00138             t, zm, zs, Z(atm->p[atm->np - 1]),
00139             Z(atm->p[atm->np - atm->np / 10]),
00140             Z(atm->p[atm->np - atm->np / 4]),
00141             Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
00142             Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00143             lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00144             atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00145             atm->lon[atm->np - atm->np / 4],
00146             atm->lon[atm->np - atm->np / 10],
00147             atm->lon[atm->np - 1],
00148             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00149             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00150             atm->lat[atm->np - atm->np / 4],
00151             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00152   }
00153
00154   /* Close file... */
00155   fclose(out);
00156
00157   /* Free... */
00158   free(atm);
00159
00160   return EXIT_SUCCESS;
00161 }
```

Here is the call graph for this function:



5.2 center.c

```
00001 /*
```

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

```
    /* 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",
            t, zm, zs, Z(atm->p[atm->np - 1]),
            Z(atm->p[atm->np - atm->np / 10]),
            Z(atm->p[atm->np - atm->np / 4]),
            Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
            Z(atm->p[atm->np / 10]), Z(atm->p[0]),
            lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
            atm->lon[atm->np / 4], atm->lon[atm->np / 2],
            atm->lon[atm->np - atm->np / 4],
            atm->lon[atm->np - atm->np / 10],
            atm->lon[atm->np - 1],
            latm, lats, atm->lat[0], atm->lat[atm->np / 10],
            atm->lat[atm->np / 4], atm->lat[atm->np / 2],
            atm->lat[atm->np - atm->np / 4],
            atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(atm);

  return EXIT_SUCCESS;
}
```

5.3 dist.c File Reference

Calculate transport deviations of trajectories.

Functions

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

5.3.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

5.3.2 Function Documentation

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

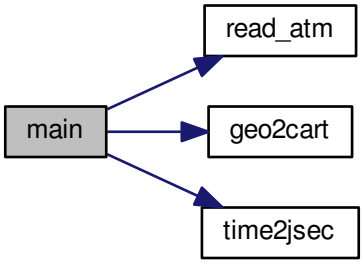
Definition at line 28 of file dist.c.

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

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

```
00182       lat2[ip] = atm2->lat[ip];
00183       p2[ip] = atm2->p[ip];
00184     }
00185
00186     /* Get indices of trajectories with maximum errors... */
00187     iph = (int) gsl_stats_max_index(dh, 1, (size_t) atm1->np);
00188     ipv = (int) gsl_stats_max_index(dv, 1, (size_t) atm1->np);
00189
00190     /* Sort distances to calculate percentiles... */
00191     gsl_sort(dh, 1, (size_t) atm1->np);
00192     gsl_sort(dv, 1, (size_t) atm1->np);
00193
00194     /* Get time from filename... */
00195     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00196     year = atoi(tstr);
00197     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00198     mon = atoi(tstr);
00199     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00200     day = atoi(tstr);
00201     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00202     hour = atoi(tstr);
00203     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00204     min = atoi(tstr);
00205     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00206
00207     /* Write output... */
00208     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d %g %g"
00209             " %g %g %g %g %g %g %g %g %g %d %g %g\n", t,
00210             ahtd / atm1->np,
00211             sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
00212             dh[0], dh[atm1->np / 10], dh[atm1->np / 4], dh[atm1->np / 2],
00213             dh[atm1->np - atm1->np / 4], dh[atm1->np - atm1->np / 10],
00214             dh[atm1->np - 1], iph, rhtd / atm1->np,
00215             sqrt(rhtd2 / atm1->np - gsl_pow_2(rhtd / atm1->np)),
00216             avtd / atm1->np,
00217             sqrt(avtd2 / atm1->np - gsl_pow_2(avtd / atm1->np)),
00218             dv[0], dv[atm1->np / 10], dv[atm1->np / 4], dv[atm1->np / 2],
00219             dv[atm1->np - atm1->np / 4], dv[atm1->np - atm1->np / 10],
00220             dv[atm1->np - 1], ipv, rvtd / atm1->np,
00221             sqrt(rvtd2 / atm1->np - gsl_pow_2(rvtd / atm1->np)));
00222   }
00223
00224   /* Close file... */
00225   fclose(out);
00226
00227   /* Free... */
00228   free(atm1);
00229   free(atm2);
00230   free(lon1);
00231   free(lat1);
00232   free(p1);
00233   free(lh1);
00234   free(lv1);
00235   free(lon2);
00236   free(lat2);
00237   free(p2);
00238   free(lh2);
00239   free(lv2);
00240   free(dh);
00241   free(dv);
00242
00243   return EXIT_SUCCESS;
00244 }
```

Here is the call graph for this function:



5.4 dist.c

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

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

```
00147       avtd2 += gsl_pow_2(dv[ip]);
00148
00149       /* Calculate relative transport deviations... */
00150       if (f > 2) {
00151
00152         /* Get trajectory lengths... */
00153         geo2cart(0, lon1[ip], lat1[ip], x0);
00154         lh1[ip] += DIST(x0, x1);
00155         lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00156
00157         geo2cart(0, lon2[ip], lat2[ip], x0);
00158         lh2[ip] += DIST(x0, x2);
00159         lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00160
00161         /* Get relative transport devations... */
00162         if (lh1[ip] + lh2[ip] > 0) {
00163           aux = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00164           rhtd += aux;
00165           rhtd2 += gsl_pow_2(aux);
00166         }
00167         if (lv1[ip] + lv2[ip] > 0) {
00168           aux =
00169             200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00170                                                             lv2[ip]);
00171           rvtd += aux;
00172           rvtd2 += gsl_pow_2(aux);
00173         }
00174       }
00175
00176       /* Save positions of air parcels... */
00177       lon1[ip] = atm1->lon[ip];
00178       lat1[ip] = atm1->lat[ip];
00179       p1[ip] = atm1->p[ip];
00180
00181       lon2[ip] = atm2->lon[ip];
00182       lat2[ip] = atm2->lat[ip];
00183       p2[ip] = atm2->p[ip];
00184     }
00185
00186     /* Get indices of trajectories with maximum errors... */
00187     iph = (int) gsl_stats_max_index(dh, 1, (size_t) atm1->np);
00188     ipv = (int) gsl_stats_max_index(dv, 1, (size_t) atm1->np);
00189
00190     /* Sort distances to calculate percentiles... */
00191     gsl_sort(dh, 1, (size_t) atm1->np);
00192     gsl_sort(dv, 1, (size_t) atm1->np);
00193
00194     /* Get time from filename... */
00195     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00196     year = atoi(tstr);
00197     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00198     mon = atoi(tstr);
00199     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00200     day = atoi(tstr);
00201     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00202     hour = atoi(tstr);
00203     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00204     min = atoi(tstr);
00205     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00206
00207     /* Write output... */
00208     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d %g %g"
00209             " %g %g %g %g %g %g %g %g %g %d %g %g\n", t,
00210             ahtd / atm1->np,
00211             sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
00212             dh[0], dh[atm1->np / 10], dh[atm1->np / 4], dh[atm1->np / 2],
00213             dh[atm1->np - atm1->np / 4], dh[atm1->np - atm1->np / 10],
00214             dh[atm1->np - 1], iph, rhtd / atm1->np,
00215             sqrt(rhtd2 / atm1->np - gsl_pow_2(rhtd / atm1->np)),
00216             avtd / atm1->np,
00217             sqrt(avtd2 / atm1->np - gsl_pow_2(avtd / atm1->np)),
00218             dv[0], dv[atm1->np / 10], dv[atm1->np / 4], dv[atm1->np / 2],
00219             dv[atm1->np - atm1->np / 4], dv[atm1->np - atm1->np / 10],
00220             dv[atm1->np - 1], ipv, rvtd / atm1->np,
00221             sqrt(rvtd2 / atm1->np - gsl_pow_2(rvtd / atm1->np)));
00222   }
00223
00224   /* Close file... */
00225   fclose(out);
00226
00227   /* Free... */
00228   free(atm1);
00229   free(atm2);
00230   free(lon1);
00231   free(lat1);
00232   free(p1);
00233   free(lh1);
```

```
00234   free(lv1);
00235   free(lon2);
00236   free(lat2);
00237   free(p2);
00238   free(lh2);
00239   free(lv2);
00240   free(dh);
00241   free(dv);
00242
00243   return EXIT_SUCCESS;
00244 }
```

5.5 extract.c File Reference

Extract single trajectory from atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.5.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

5.5.2 Function Documentation

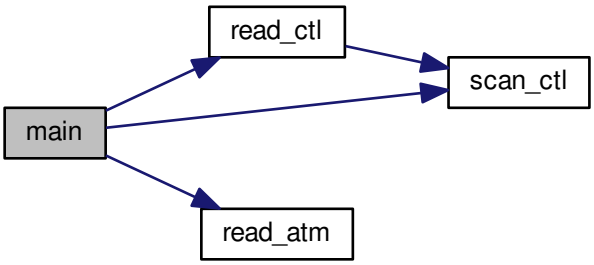
5.5.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 28 of file extract.c.

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

```
00063   for (iq = 0; iq < ctl.nq; iq++)
00064     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00065             ctl.qnt_unit[iq]);
00066   fprintf(out, "\n");
00067
00068   /* Loop over files... */
00069   for (f = 3; f < argc; f++) {
00070
00071     /* Read atmopheric data... */
00072     if (!(in = fopen(argv[f], "r")))
00073       continue;
00074     else
00075       fclose(in);
00076     read_atm(argv[f], &ctl, atm);
00077
00078     /* Write data... */
00079     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081     for (iq = 0; iq < ctl.nq; iq++) {
00082       fprintf(out, " ");
00083       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084     }
00085     fprintf(out, "\n");
00086   }
00087
00088   /* Close file... */
00089   fclose(out);
00090
00091   /* Free... */
00092   free(atm);
00093
00094   return EXIT_SUCCESS;
00095 }
```

Here is the call graph for this function:



5.6 extract.c

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

```
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029   int argc,
00030   char *argv[]) {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm;
00035
00036   FILE *in, *out;
00037
00038   int f, ip, iq;
00039
00040   /* Allocate... */
00041   ALLOC(atm, atm_t, 1);
00042
00043   /* Check arguments... */
00044   if (argc < 4)
00045     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00046
00047   /* Read control parameters... */
00048   read_ctl(argv[1], argc, argv, &ctl);
00049   ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00050
00051   /* Write info... */
00052   printf("Write trajectory data: %s\n", argv[2]);
00053
00054   /* Create output file... */
00055   if (!(out = fopen(argv[2], "w")))
00056     ERRMSG("Cannot create file!");
00057
00058   /* Write header... */
00059   fprintf(out,
00060           "# $1 = time [s]\n"
00061           "# $2 = altitude [km]\n"
00062           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00063   for (iq = 0; iq < ctl.nq; iq++)
00064     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00065             ctl.qnt_unit[iq]);
00066   fprintf(out, "\n");
00067
00068   /* Loop over files... */
00069   for (f = 3; f < argc; f++) {
00070
00071     /* Read atmopheric data... */
00072     if (!(in = fopen(argv[f], "r")))
00073       continue;
00074     else
00075       fclose(in);
00076     read_atm(argv[f], &ctl, atm);
00077
00078     /* Write data... */
00079     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081     for (iq = 0; iq < ctl.nq; iq++) {
00082       fprintf(out, " ");
00083       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084     }
00085     fprintf(out, "\n");
00086   }
00087
00088   /* Close file... */
00089   fclose(out);
00090
00091   /* Free... */
00092   free(atm);
00093
00094   return EXIT_SUCCESS;
00095 }
```

5.7 init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

- int main (int argc, char ∗argv[ ])

5.7.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

5.7.2 Function Documentation

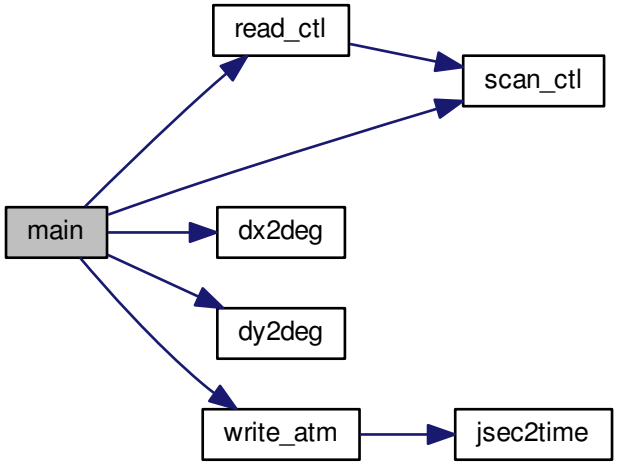
5.7.2.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 27 of file init.c.

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

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

Here is the call graph for this function:



5.8 init.c

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

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

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

5.9 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

- int main (int argc, char ∗argv[ ])

5.9.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

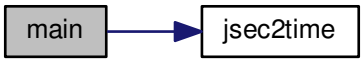
5.9.2 Function Documentation

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

Definition at line 27 of file jsec2time.c.

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

Here is the call graph for this function:



5.10 jsec2time.c

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

5.11 libtrac.c File Reference

MPTRAC library definitions.

Functions

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

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

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.

5.11.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

5.11.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

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

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

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

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

5.11.2.3 double deg2dy (double *dlat*)

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

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

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

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

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

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

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

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

5.11.2.6 double dy2deg (double *dy*)

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

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

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

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

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

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

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

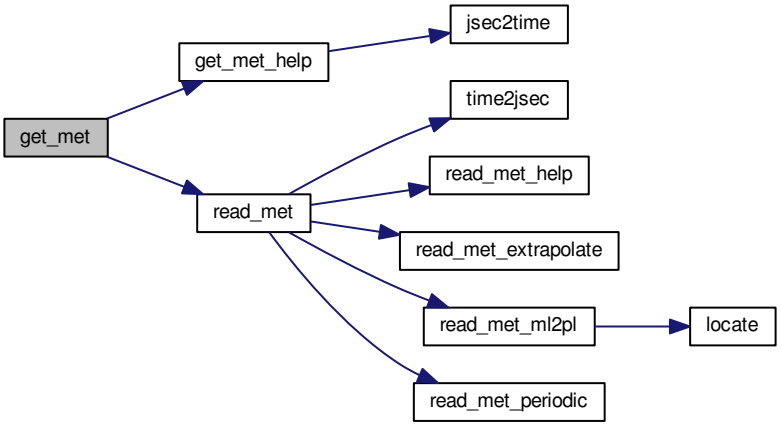
5.11.2.9 void get_met (ctl_t * *ctl*, char * *metbase*, double *t*, met_t * *met0*, met_t * *met1*)

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

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

Here is the call graph for this function:



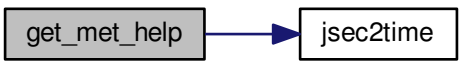
5.11.2.10 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

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

Here is the call graph for this function:



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

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

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

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

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

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

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

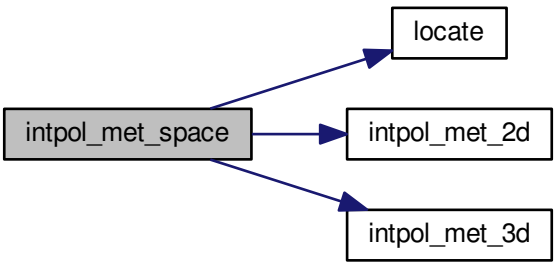
Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

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

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

Here is the call graph for this function:



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

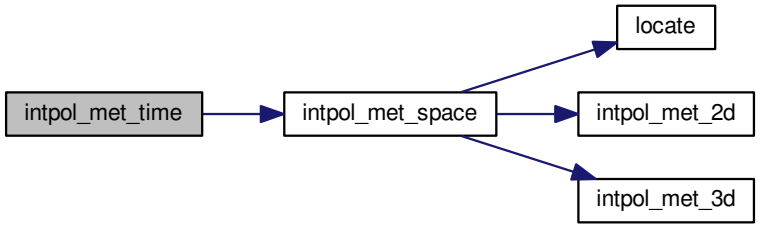
Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

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

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

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

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

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

Find array index.

Definition at line 376 of file libtrac.c.

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

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

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

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

Read control parameters.

Definition at line 458 of file libtrac.c.

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

```
00539       sprintf(ctl->qnt_unit[iq], "1");
00540     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00541       ctl->qnt_theta = iq;
00542       sprintf(ctl->qnt_unit[iq], "K");
00543     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00544       ctl->qnt_pv = iq;
00545       sprintf(ctl->qnt_unit[iq], "PVU");
00546     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00547       ctl->qnt_tice = iq;
00548       sprintf(ctl->qnt_unit[iq], "K");
00549     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00550       ctl->qnt_tsts = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00553       ctl->qnt_tnat = iq;
00554       sprintf(ctl->qnt_unit[iq], "K");
00555     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00556       ctl->qnt_gw_var = iq;
00557       sprintf(ctl->qnt_unit[iq], "K^2");
00558     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00559       ctl->qnt_stat = iq;
00560       sprintf(ctl->qnt_unit[iq], "-");
00561     } else
00562       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00563   }
00564
00565   /* Time steps of simulation... */
00566   ctl->direction =
00567     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00568   if (ctl->direction != -1 && ctl->direction != 1)
00569     ERRMSG("Set DIRECTION to -1 or 1!");
00570   ctl->t_start =
00571     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00572   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00573   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00574
00575   /* Meteorological data... */
00576   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00577   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00578   if (ctl->met_np > EP)
00579     ERRMSG("Too many levels!");
00580   for (ip = 0; ip < ctl->met_np; ip++)
00581     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00582
00583   /* Isosurface parameters... */
00584   ctl->isosurf
00585     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00586   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00587
00588   /* Diffusion parameters... */
00589   ctl->turb_dx_trop
00590     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00591   ctl->turb_dx_strat
00592     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00593   ctl->turb_dz_trop
00594     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00595   ctl->turb_dz_strat
00596     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00597   ctl->turb_meso =
00598     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00599
00600   /* Life time of particles... */
00601   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00602   ctl->tdec_strat =
00603     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00604
00605   /* PSC analysis... */
00606   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00607   ctl->psc_hno3 =
00608     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00609
00610   /* Gravity wave analysis... */
00611   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
00612
00613   /* Output of atmospheric data... */
00614   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00615   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00616   ctl->atm_dt_out =
00617     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00618   ctl->atm_filter =
00619     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00620
00621   /* Output of CSI data... */
00622   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
```

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

Here is the call graph for this function:

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

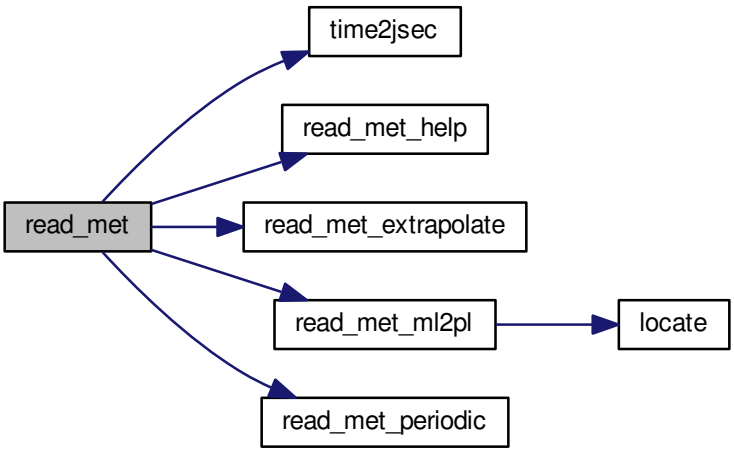
Read meteorological data file.

Definition at line 703 of file libtrac.c.

```
00706               {
00707
00708   char tstr[10];
00709
00710   static float help[EX * EY];
00711
00712   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00713
00714   size_t np, nx, ny;
00715
00716   /* Write info... */
00717   printf("Read meteorological data: %s\n", filename);
00718
00719   /* Get time from filename... */
00720   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00721   year = atoi(tstr);
00722   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00723   mon = atoi(tstr);
00724   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00725   day = atoi(tstr);
00726   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00727   hour = atoi(tstr);
00728   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00729
00730   /* Open netCDF file... */
00731   NC(nc_open(filename, NC_NOWRITE, &ncid));
00732
00733   /* Get dimensions... */
00734   NC(nc_inq_dimid(ncid, "lon", &dimid));
00735   NC(nc_inq_dimlen(ncid, dimid, &nx));
00736   if (nx > EX)
00737     ERRMSG("Too many longitudes!");
00738
00739   NC(nc_inq_dimid(ncid, "lat", &dimid));
00740   NC(nc_inq_dimlen(ncid, dimid, &ny));
00741   if (ny > EY)
00742     ERRMSG("Too many latitudes!");
00743
00744   NC(nc_inq_dimid(ncid, "lev", &dimid));
00745   NC(nc_inq_dimlen(ncid, dimid, &np));
00746   if (np > EP)
00747     ERRMSG("Too many levels!");
00748
00749   /* Store dimensions... */
00750   met->np = (int) np;
00751   met->nx = (int) nx;
00752   met->ny = (int) ny;
00753
00754   /* Get horizontal grid... */
00755   NC(nc_inq_varid(ncid, "lon", &varid));
00756   NC(nc_get_var_double(ncid, varid, met->lon));
00757   NC(nc_inq_varid(ncid, "lat", &varid));
00758   NC(nc_get_var_double(ncid, varid, met->lat));
00759
00760   /* Read meteorological data... */
00761   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00762   read_met_help(ncid, "u", "U", met, met->u, 1.0);
```

```
00763   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00764   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00765   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00766   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00767
00768   /* Meteo data on pressure levels... */
00769   if (ctl->met_np <= 0) {
00770
00771     /* Read pressure levels from file... */
00772     NC(nc_inq_varid(ncid, "lev", &varid));
00773     NC(nc_get_var_double(ncid, varid, met->p));
00774     for (ip = 0; ip < met->np; ip++)
00775       met->p[ip] /= 100.;
00776
00777     /* Extrapolate data for lower boundary... */
00778     read_met_extrapolate(met);
00779   }
00780
00781   /* Meteo data on model levels... */
00782   else {
00783
00784     /* Read pressure data from file... */
00785     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00786
00787     /* Interpolate from model levels to pressure levels... */
00788     read_met_ml2pl(ctl, met, met->t);
00789     read_met_ml2pl(ctl, met, met->u);
00790     read_met_ml2pl(ctl, met, met->v);
00791     read_met_ml2pl(ctl, met, met->w);
00792     read_met_ml2pl(ctl, met, met->h2o);
00793     read_met_ml2pl(ctl, met, met->o3);
00794
00795     /* Set pressure levels... */
00796     met->np = ctl->met_np;
00797     for (ip = 0; ip < met->np; ip++)
00798       met->p[ip] = ctl->met_p[ip];
00799   }
00800
00801   /* Check ordering of pressure levels... */
00802   for (ip = 1; ip < met->np; ip++)
00803     if (met->p[ip - 1] < met->p[ip])
00804       ERRMSG("Pressure levels must be descending!");
00805
00806   /* Read surface pressure... */
00807   if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00808     NC(nc_get_var_float(ncid, varid, help));
00809     for (iy = 0; iy < met->ny; iy++)
00810       for (ix = 0; ix < met->nx; ix++)
00811         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00812   } else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00813     NC(nc_get_var_float(ncid, varid, help));
00814     for (iy = 0; iy < met->ny; iy++)
00815       for (ix = 0; ix < met->nx; ix++)
00816         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00817   } else
00818     for (ix = 0; ix < met->nx; ix++)
00819       for (iy = 0; iy < met->ny; iy++)
00820         met->ps[ix][iy] = met->p[0];
00821
00822   /* Create periodic boundary conditions... */
00823   read_met_periodic(met);
00824
00825   /* Close file... */
00826   NC(nc_close(ncid));
00827 }
```

Here is the call graph for this function:



5.11.2.20 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 831 of file libtrac.c.

```
00832                {
00833
00834   int ip, ip0, ix, iy;
00835
00836   /* Loop over columns... */
00837   for (ix = 0; ix < met->nx; ix++)
00838     for (iy = 0; iy < met->ny; iy++) {
00839
00840       /* Find lowest valid data point... */
00841       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00842         if (!gsl_finite(met->t[ix][iy][ip0])
00843             || !gsl_finite(met->u[ix][iy][ip0])
00844             || !gsl_finite(met->v[ix][iy][ip0])
00845             || !gsl_finite(met->w[ix][iy][ip0]))
00846           break;
00847
00848       /* Extrapolate... */
00849       for (ip = ip0; ip >= 0; ip--) {
00850         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00851         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00852         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00853         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00854         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00855         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00856       }
00857     }
00858 }
```

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

Read and convert variable from meteorological data file.

Definition at line 862 of file libtrac.c.

```
00868             {
00869
00870   static float help[EX * EY * EP];
00871
00872   int ip, ix, iy, n = 0, varid;
00873
00874   /* Check if variable exists... */
00875   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00876     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00877       return;
00878
00879   /* Read data... */
00880   NC(nc_get_var_float(ncid, varid, help));
00881
00882   /* Copy and check data... */
00883   for (ip = 0; ip < met->np; ip++)
00884     for (iy = 0; iy < met->ny; iy++)
00885       for (ix = 0; ix < met->nx; ix++) {
00886         dest[ix][iy][ip] = scl * help[n++];
00887         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00888           dest[ix][iy][ip] = GSL_NAN;
00889       }
00890 }
```

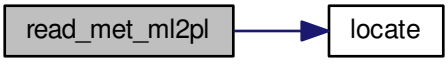
5.11.2.22 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 894 of file libtrac.c.

```
00897                         {
00898
00899   double aux[EP], p[EP], pt;
00900
00901   int ip, ip2, ix, iy;
00902
00903   /* Loop over columns... */
00904   for (ix = 0; ix < met->nx; ix++)
00905     for (iy = 0; iy < met->ny; iy++) {
00906
00907       /* Copy pressure profile... */
00908       for (ip = 0; ip < met->np; ip++)
00909         p[ip] = met->pl[ix][iy][ip];
00910
00911       /* Interpolate... */
00912       for (ip = 0; ip < ctl->met_np; ip++) {
00913         pt = ctl->met_p[ip];
00914         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00915           pt = p[0];
00916         else if ((pt > p[met->np - 1] && p[1] > p[0])
00917                  || (pt < p[met->np - 1] && p[1] < p[0]))
00918           pt = p[met->np - 1];
00919         ip2 = locate(p, met->np, pt);
00920         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00921                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00922       }
00923
00924       /* Copy data... */
00925       for (ip = 0; ip < ctl->met_np; ip++)
00926         var[ix][iy][ip] = (float) aux[ip];
00927     }
00928 }
```

Here is the call graph for this function:



5.11.2.23 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 932 of file libtrac.c.

```
00933               {
00934
00935   int ip, iy;
00936
00937   /* Check longitudes... */
00938   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00939              + met->lon[1] - met->lon[0] - 360) < 0.01))
00940     return;
00941
00942   /* Increase longitude counter... */
00943   if ((++met->nx) > EX)
00944     ERRMSG("Cannot create periodic boundary conditions!");
00945
00946   /* Set longitude... */
00947   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00948
00949   /* Loop over latitudes and pressure levels... */
00950   for (iy = 0; iy < met->ny; iy++)
00951     for (ip = 0; ip < met->np; ip++) {
00952       met->ps[met->nx - 1][iy] = met->ps[0][iy];
00953       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
00954       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00955       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00956       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00957       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00958       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00959     }
00960 }
```

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

Read a control parameter from file or command line.

Definition at line 964 of file libtrac.c.

```
00971               {
00972
00973   FILE *in = NULL;
00974
00975   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00976     msg[LEN], rvarname[LEN], rval[LEN];
00977
00978   int contain = 0, i;
00979
00980   /* Open file... */
00981   if (filename[strlen(filename) - 1] != '-')
00982     if (!(in = fopen(filename, "r")))
00983       ERRMSG("Cannot open file!");
00984
00985   /* Set full variable name... */
00986   if (arridx >= 0) {
00987     sprintf(fullname1, "%s[%d]", varname, arridx);
00988     sprintf(fullname2, "%s[*]", varname);
00989   } else {
00990     sprintf(fullname1, "%s", varname);
00991     sprintf(fullname2, "%s", varname);
00992   }
00993
00994   /* Read data... */
00995   if (in != NULL)
00996     while (fgets(line, LEN, in))
00997       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00998         if (strcasecmp(rvarname, fullname1) == 0 ||
00999             strcasecmp(rvarname, fullname2) == 0) {
01000           contain = 1;
01001           break;
01002         }
01003   for (i = 1; i < argc - 1; i++)
```

```
01004     if (strcasecmp(argv[i], fullname1) == 0 ||
01005         strcasecmp(argv[i], fullname2) == 0) {
01006       sprintf(rval, "%s", argv[i + 1]);
01007       contain = 1;
01008       break;
01009     }
01010
01011   /* Close file... */
01012   if (in != NULL)
01013     fclose(in);
01014
01015   /* Check for missing variables... */
01016   if (!contain) {
01017     if (strlen(defvalue) > 0)
01018       sprintf(rval, "%s", defvalue);
01019     else {
01020       sprintf(msg, "Missing variable %s!\n", fullname1);
01021       ERRMSG(msg);
01022     }
01023   }
01024
01025   /* Write info... */
01026   printf("%s = %s\n", fullname1, rval);
01027
01028   /* Return values... */
01029   if (value != NULL)
01030     sprintf(value, "%s", rval);
01031   return atof(rval);
01032 }
```

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

Convert date to seconds.

Definition at line 1036 of file libtrac.c.

```
01044                 {
01045
01046   struct tm t0, t1;
01047
01048   t0.tm_year = 100;
01049   t0.tm_mon = 0;
01050   t0.tm_mday = 1;
01051   t0.tm_hour = 0;
01052   t0.tm_min = 0;
01053   t0.tm_sec = 0;
01054
01055   t1.tm_year = year - 1900;
01056   t1.tm_mon = mon - 1;
01057   t1.tm_mday = day;
01058   t1.tm_hour = hour;
01059   t1.tm_min = min;
01060   t1.tm_sec = sec;
01061
01062   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01063 }
```

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

Measure wall-clock time.

Definition at line 1067 of file libtrac.c.

```
01070             {
01071
01072   static double starttime[NTIMER], runtime[NTIMER];
01073
01074   /* Check id... */
01075   if (id < 0 || id >= NTIMER)
01076     ERRMSG("Too many timers!");
01077
01078   /* Start timer... */
01079   if (mode == 1) {
01080     if (starttime[id] <= 0)
```

```
01081       starttime[id] = omp_get_wtime();
01082     else
01083       ERRMSG("Timer already started!");
01084   }
01085
01086   /* Stop timer... */
01087   else if (mode == 2) {
01088     if (starttime[id] > 0) {
01089       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01090       starttime[id] = -1;
01091     } else
01092       ERRMSG("Timer not started!");
01093   }
01094
01095   /* Print timer... */
01096   else if (mode == 3)
01097     printf("%s = %g s\n", name, runtime[id]);
01098 }
```

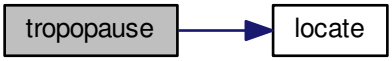
5.11.2.27 double tropopause (double *t*, double *lat*)

Definition at line 1102 of file libtrac.c.

```
01104               {
01105
01106   static double doys[12]
01107   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01108
01109   static double lats[73]
01110     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01111     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01112     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01113     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01114     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01115     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01116     75, 77.5, 80, 82.5, 85, 87.5, 90
01117   };
01118
01119   static double tps[12][73]
01120     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01121          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01122          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01123          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01124          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01125          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01126          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01127          275.3, 275.6, 275.4, 274.1, 273.5},
01128   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01129    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01130    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01131    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01132    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01133    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01134    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01135    287.5, 286.2, 285.8},
01136   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01137    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01138    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01139    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01140    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01141    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01142    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01143    304.3, 304.9, 306, 306.6, 306.2, 306},
01144   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01145    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01146    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01147    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01148    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01149    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01150    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01151    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01152   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01153    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01154    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01155    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01156    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01157    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01158    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01159    325.3, 325.8, 325.8},
01160   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01161    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
```

```
01162     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01163     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01164     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01165     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01166     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01167     308.5, 312.2, 313.1, 313.3},
01168    {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01169     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01170     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01171     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01172     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01173     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01174     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01175     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01176    {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01177     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01178     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01179     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01180     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01181     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01182     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01183     278.2, 282.6, 287.4, 290.9, 292.5, 293},
01184    {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01185     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01186     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01187     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01188     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01189     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01190     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01191     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01192    {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01193     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01194     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01195     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01196     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01197     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01198     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01199     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01200     305.1},
01201    {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01202     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01203     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01204     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01205     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01206     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01207     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01208     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01209    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01210     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01211     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01212     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01213     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01214     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01215     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01216     281.7, 281.1, 281.2}
01217    };
01218
01219    double doy, p0, p1, pt;
01220
01221    int imon, ilat;
01222
01223    /* Get day of year... */
01224    doy = fmod(t / 86400., 365.25);
01225    while (doy < 0)
01226      doy += 365.25;
01227
01228    /* Get indices... */
01229    imon = locate(doys, 12, doy);
01230    ilat = locate(lats, 73, lat);
01231
01232    /* Get tropopause pressure... */
01233    p0 = LIN(lats[ilat], tps[imon][ilat],
01234             lats[ilat + 1], tps[imon][ilat + 1], lat);
01235    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01236             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01237    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01238
01239    /* Return tropopause pressure... */
01240    return pt;
01241  }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1245 of file libtrac.c.

```
01249            {
01250
01251   FILE *in, *out;
01252
01253   char line[LEN];
01254
01255   double r, t0, t1;
01256
01257   int ip, iq, year, mon, day, hour, min, sec;
01258
01259   /* Set time interval for output... */
01260   t0 = t - 0.5 * ctl->dt_mod;
01261   t1 = t + 0.5 * ctl->dt_mod;
01262
01263   /* Check if gnuplot output is requested... */
01264   if (ctl->atm_gpfile[0] != '-') {
01265
01266     /* Write info... */
01267     printf("Plot atmospheric data: %s.png\n", filename);
01268
01269     /* Create gnuplot pipe... */
01270     if (!(out = popen("gnuplot", "w")))
01271       ERRMSG("Cannot create pipe to gnuplot!");
01272
01273     /* Set plot filename... */
01274     fprintf(out, "set out \"%s.png\"\n", filename);
01275
01276     /* Set time string... */
01277     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01278     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01279             year, mon, day, hour, min);
01280
01281     /* Dump gnuplot file to pipe... */
01282     if (!(in = fopen(ctl->atm_gpfile, "r")))
01283       ERRMSG("Cannot open file!");
01284     while (fgets(line, LEN, in))
01285       fprintf(out, "%s", line);
01286     fclose(in);
01287   }
01288
01289   else {
01290
01291     /* Write info... */
01292     printf("Write atmospheric data: %s\n", filename);
01293
01294     /* Create file... */
01295     if (!(out = fopen(filename, "w")))
01296       ERRMSG("Cannot create file!");
01297   }
01298
01299   /* Write header... */
01300   fprintf(out,
01301           "# $1 = time [s]\n"
01302           "# $2 = altitude [km]\n"
01303           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01304   for (iq = 0; iq < ctl->nq; iq++)
01305     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
```

```
01306             ctl->qnt_unit[iq]);
01307   fprintf(out, "\n");
01308
01309   /* Write data... */
01310   for (ip = 0; ip < atm->np; ip++) {
01311
01312     /* Check time... */
01313     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01314       continue;
01315
01316     /* Write output... */
01317     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01318             atm->lon[ip], atm->lat[ip]);
01319     for (iq = 0; iq < ctl->nq; iq++) {
01320       fprintf(out, " ");
01321       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01322     }
01323     fprintf(out, "\n");
01324   }
01325
01326   /* Close file... */
01327   fclose(out);
01328 }
```

Here is the call graph for this function:

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

Write CSI data.

Definition at line 1332 of file libtrac.c.

```
01336             {
01337
01338   static FILE *in, *out;
01339
01340   static char line[LEN];
01341
01342   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01343     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01344
01345   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01346
01347   /* Init... */
01348   if (!init) {
01349     init = 1;
01350
01351     /* Check quantity index for mass... */
01352     if (ctl->qnt_m < 0)
01353       ERRMSG("Need quantity mass to analyze CSI!");
01354
01355     /* Open observation data file... */
01356     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01357     if (!(in = fopen(ctl->csi_obsfile, "r")))
01358       ERRMSG("Cannot open file!");
01359
01360     /* Create new file... */
01361     printf("Write CSI data: %s\n", filename);
01362     if (!(out = fopen(filename, "w")))
01363       ERRMSG("Cannot create file!");
01364
01365     /* Write header... */
```

```
01366     fprintf(out,
01367             "# $1 = time [s]\n"
01368             "# $2 = number of hits (cx)\n"
01369             "# $3 = number of misses (cy)\n"
01370             "# $4 = number of false alarms (cz)\n"
01371             "# $5 = number of observations (cx + cy)\n"
01372             "# $6 = number of forecasts (cx + cz)\n"
01373             "# $7 = bias (forecasts/observations) [%%]\n"
01374             "# $8 = probability of detection (POD) [%%]\n"
01375             "# $9 = false alarm rate (FAR) [%%]\n"
01376             "# $10 = critical success index (CSI) [%%]\n\n");
01377   }
01378
01379   /* Set time interval... */
01380   t0 = t - 0.5 * ctl->dt_mod;
01381   t1 = t + 0.5 * ctl->dt_mod;
01382
01383   /* Initialize grid cells... */
01384   for (ix = 0; ix < ctl->csi_nx; ix++)
01385     for (iy = 0; iy < ctl->csi_ny; iy++)
01386       for (iz = 0; iz < ctl->csi_nz; iz++)
01387         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01388
01389   /* Read data... */
01390   while (fgets(line, LEN, in)) {
01391
01392     /* Read data... */
01393     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01394         5)
01395       continue;
01396
01397     /* Check time... */
01398     if (rt < t0)
01399       continue;
01400     if (rt > t1)
01401       break;
01402
01403     /* Calculate indices... */
01404     ix = (int) ((rlon - ctl->csi_lon0)
01405                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01406     iy = (int) ((rlat - ctl->csi_lat0)
01407                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01408     iz = (int) ((rz - ctl->csi_z0)
01409                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01410
01411     /* Check indices... */
01412     if (ix < 0 || ix >= ctl->csi_nx ||
01413         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01414       continue;
01415
01416     /* Get mean observation index... */
01417     obsmean[ix][iy][iz] += robs;
01418     obscount[ix][iy][iz]++;
01419   }
01420
01421   /* Analyze model data... */
01422   for (ip = 0; ip < atm->np; ip++) {
01423
01424     /* Check time... */
01425     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01426       continue;
01427
01428     /* Get indices... */
01429     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01430                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01431     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01432                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01433     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01434                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01435
01436     /* Check indices... */
01437     if (ix < 0 || ix >= ctl->csi_nx ||
01438         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01439       continue;
01440
01441     /* Get total mass in grid cell... */
01442     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01443   }
01444
01445   /* Analyze all grid cells... */
01446   for (ix = 0; ix < ctl->csi_nx; ix++)
01447     for (iy = 0; iy < ctl->csi_ny; iy++)
01448       for (iz = 0; iz < ctl->csi_nz; iz++) {
01449
01450         /* Calculate mean observation index... */
01451         if (obscount[ix][iy][iz] > 0)
01452           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
```

```
01453
01454         /* Calculate column density... */
01455         if (modmean[ix][iy][iz] > 0) {
01456           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01457           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01458           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01459           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01460             * cos(lat * M_PI / 180.);
01461           modmean[ix][iy][iz] /= (1e6 * area);
01462         }
01463
01464         /* Calculate CSI... */
01465         if (obscount[ix][iy][iz] > 0) {
01466           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01467               modmean[ix][iy][iz] >= ctl->csi_modmin)
01468             cx++;
01469           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01470                    modmean[ix][iy][iz] < ctl->csi_modmin)
01471             cy++;
01472           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01473                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01474             cz++;
01475         }
01476       }
01477
01478   /* Write output... */
01479   if (fmod(t, ctl->csi_dt_out) == 0) {
01480
01481     /* Write... */
01482     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01483             t, cx, cy, cz, cx + cy, cx + cz,
01484             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01485             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01486             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01487             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01488
01489     /* Set counters to zero... */
01490     cx = cy = cz = 0;
01491   }
01492
01493   /* Close file... */
01494   if (t == ctl->t_stop)
01495     fclose(out);
01496 }
```

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

Write ensemble data.

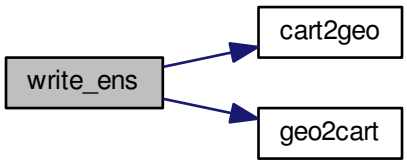
Definition at line 1500 of file libtrac.c.

```
01504             {
01505
01506   static FILE *out;
01507
01508   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01509     t0, t1, x[NENS][3], xm[3];
01510
01511   static int init, ip, iq;
01512
01513   static size_t i, n;
01514
01515   /* Init... */
01516   if (!init) {
01517     init = 1;
01518
01519     /* Check quantities... */
01520     if (ctl->qnt_ens < 0)
01521       ERRMSG("Missing ensemble IDs!");
01522
01523     /* Create new file... */
01524     printf("Write ensemble data: %s\n", filename);
01525     if (!(out = fopen(filename, "w")))
01526       ERRMSG("Cannot create file!");
01527
01528     /* Write header... */
01529     fprintf(out,
01530             "# $1 = time [s]\n"
01531             "# $2 = altitude [km]\n"
01532             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
```

```
01533     for (iq = 0; iq < ctl->nq; iq++)
01534       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01535               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01536     for (iq = 0; iq < ctl->nq; iq++)
01537       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01538               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01539     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01540   }
01541
01542   /* Set time interval... */
01543   t0 = t - 0.5 * ctl->dt_mod;
01544   t1 = t + 0.5 * ctl->dt_mod;
01545
01546   /* Init... */
01547   ens = GSL_NAN;
01548   n = 0;
01549
01550   /* Loop over air parcels... */
01551   for (ip = 0; ip < atm->np; ip++) {
01552
01553     /* Check time... */
01554     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01555       continue;
01556
01557     /* Check ensemble id... */
01558     if (atm->q[ctl->qnt_ens][ip] != ens) {
01559
01560       /* Write results... */
01561       if (n > 0) {
01562
01563         /* Get mean position... */
01564         xm[0] = xm[1] = xm[2] = 0;
01565         for (i = 0; i < n; i++) {
01566           xm[0] += x[i][0] / (double) n;
01567           xm[1] += x[i][1] / (double) n;
01568           xm[2] += x[i][2] / (double) n;
01569         }
01570         cart2geo(xm, &dummy, &lon, &lat);
01571         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01572                 lat);
01573
01574         /* Get quantity statistics... */
01575         for (iq = 0; iq < ctl->nq; iq++) {
01576           fprintf(out, " ");
01577           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01578         }
01579         for (iq = 0; iq < ctl->nq; iq++) {
01580           fprintf(out, " ");
01581           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01582         }
01583         fprintf(out, " %lu\n", n);
01584       }
01585
01586       /* Init new ensemble... */
01587       ens = atm->q[ctl->qnt_ens][ip];
01588       n = 0;
01589     }
01590
01591     /* Save data... */
01592     p[n] = atm->p[ip];
01593     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01594     for (iq = 0; iq < ctl->nq; iq++)
01595       q[iq][n] = atm->q[iq][ip];
01596     if ((++n) >= NENS)
01597       ERRMSG("Too many data points!");
01598   }
01599
01600   /* Write results... */
01601   if (n > 0) {
01602
01603     /* Get mean position... */
01604     xm[0] = xm[1] = xm[2] = 0;
01605     for (i = 0; i < n; i++) {
01606       xm[0] += x[i][0] / (double) n;
01607       xm[1] += x[i][1] / (double) n;
01608       xm[2] += x[i][2] / (double) n;
01609     }
01610     cart2geo(xm, &dummy, &lon, &lat);
01611     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01612
01613     /* Get quantity statistics... */
01614     for (iq = 0; iq < ctl->nq; iq++) {
01615       fprintf(out, " ");
01616       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01617     }
01618     for (iq = 0; iq < ctl->nq; iq++) {
01619       fprintf(out, " ");
```

```
01620       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01621     }
01622     fprintf(out, " %lu\n", n);
01623   }
01624
01625   /* Close file... */
01626   if (t == ctl->t_stop)
01627     fclose(out);
01628 }
```

Here is the call graph for this function:



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

Write gridded data.

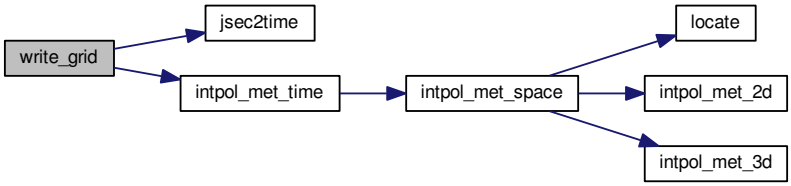
Definition at line 1632 of file libtrac.c.

```
01638             {
01639
01640   FILE *in, *out;
01641
01642   char line[LEN];
01643
01644   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01645     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01646
01647   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01648
01649   /* Check dimensions... */
01650   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01651     ERRMSG("Grid dimensions too large!");
01652
01653   /* Check quantity index for mass... */
01654   if (ctl->qnt_m < 0)
01655     ERRMSG("Need quantity mass to write grid data!");
01656
01657   /* Set time interval for output... */
01658   t0 = t - 0.5 * ctl->dt_mod;
01659   t1 = t + 0.5 * ctl->dt_mod;
01660
01661   /* Set grid box size... */
01662   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01663   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01664   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01665
01666   /* Initialize grid... */
01667   for (ix = 0; ix < ctl->grid_nx; ix++)
01668     for (iy = 0; iy < ctl->grid_ny; iy++)
01669       for (iz = 0; iz < ctl->grid_nz; iz++)
01670         grid_m[ix][iy][iz] = 0;
01671
01672   /* Average data... */
01673   for (ip = 0; ip < atm->np; ip++)
01674     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01675
01676       /* Get index... */
01677       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
```

```
01678       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01679       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01680
01681       /* Check indices... */
01682       if (ix < 0 || ix >= ctl->grid_nx ||
01683           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01684         continue;
01685
01686       /* Add mass... */
01687       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01688     }
01689
01690   /* Check if gnuplot output is requested... */
01691   if (ctl->grid_gpfile[0] != '-') {
01692
01693     /* Write info... */
01694     printf("Plot grid data: %s.png\n", filename);
01695
01696     /* Create gnuplot pipe... */
01697     if (!(out = popen("gnuplot", "w")))
01698       ERRMSG("Cannot create pipe to gnuplot!");
01699
01700     /* Set plot filename... */
01701     fprintf(out, "set out \"%s.png\"\n", filename);
01702
01703     /* Set time string... */
01704     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01705     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01706             year, mon, day, hour, min);
01707
01708     /* Dump gnuplot file to pipe... */
01709     if (!(in = fopen(ctl->grid_gpfile, "r")))
01710       ERRMSG("Cannot open file!");
01711     while (fgets(line, LEN, in))
01712       fprintf(out, "%s", line);
01713     fclose(in);
01714   }
01715
01716   else {
01717
01718     /* Write info... */
01719     printf("Write grid data: %s\n", filename);
01720
01721     /* Create file... */
01722     if (!(out = fopen(filename, "w")))
01723       ERRMSG("Cannot create file!");
01724   }
01725
01726   /* Write header... */
01727   fprintf(out,
01728           "# $1 = time [s]\n"
01729           "# $2 = altitude [km]\n"
01730           "# $3 = longitude [deg]\n"
01731           "# $4 = latitude [deg]\n"
01732           "# $5 = surface area [km^2]\n"
01733           "# $6 = layer width [km]\n"
01734           "# $7 = temperature [K]\n"
01735           "# $8 = column density [kg/m^2]\n"
01736           "# $9 = mass mixing ratio [1]\n\n");
01737
01738   /* Write data... */
01739   for (ix = 0; ix < ctl->grid_nx; ix++) {
01740     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01741       fprintf(out, "\n");
01742     for (iy = 0; iy < ctl->grid_ny; iy++) {
01743       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01744         fprintf(out, "\n");
01745       for (iz = 0; iz < ctl->grid_nz; iz++)
01746         if (!ctl->grid_sparse
01747             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01748
01749           /* Set coordinates... */
01750           z = ctl->grid_z0 + dz * (iz + 0.5);
01751           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01752           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01753
01754           /* Get pressure and temperature... */
01755           press = P(z);
01756           intpol_met_time(met0, met1, t, press, lon, lat,
01757                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01758
01759           /* Calculate surface area... */
01760           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01761             * cos(lat * M_PI / 180.);
01762
01763           /* Calculate column density... */
01764           cd = grid_m[ix][iy][iz] / (1e6 * area);
```

```
01765
01766           /* Calculate mass mixing ratio... */
01767           rho_air = 100. * press / (287.058 * temp);
01768           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01769
01770           /* Write output... */
01771           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01772                   t, z, lon, lat, area, dz, temp, cd, mmr);
01773         }
01774     }
01775   }
01776
01777   /* Close file... */
01778   fclose(out);
01779 }
```

Here is the call graph for this function:



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

Write profile data.

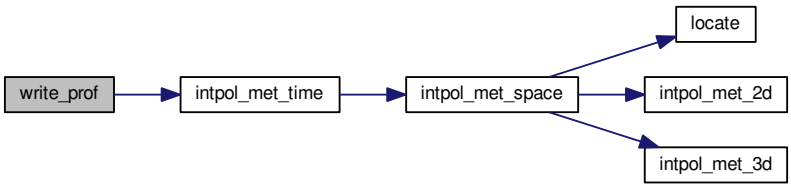
Definition at line 1783 of file libtrac.c.

```
01789           {
01790
01791   static FILE *in, *out;
01792
01793   static char line[LEN];
01794
01795   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01796     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01797     press, temp, rho_air, mmr, h2o, o3;
01798
01799   static int init, obscount[GX][GY], ip, ix, iy, iz;
01800
01801   /* Init... */
01802   if (!init) {
01803     init = 1;
01804
01805     /* Check quantity index for mass... */
01806     if (ctl->qnt_m < 0)
01807       ERRMSG("Need quantity mass!");
01808
01809     /* Check dimensions... */
01810     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01811       ERRMSG("Grid dimensions too large!");
01812
01813     /* Open observation data file... */
01814     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01815     if (!(in = fopen(ctl->prof_obsfile, "r")))
01816       ERRMSG("Cannot open file!");
01817
01818     /* Create new file... */
01819     printf("Write profile data: %s\n", filename);
01820     if (!(out = fopen(filename, "w")))
01821       ERRMSG("Cannot create file!");
01822
01823     /* Write header... */
01824     fprintf(out,
```

```
01825             "# $1  = time [s]\n"
01826             "# $2  = altitude [km]\n"
01827             "# $3  = longitude [deg]\n"
01828             "# $4  = latitude [deg]\n"
01829             "# $5  = pressure [hPa]\n"
01830             "# $6  = temperature [K]\n"
01831             "# $7  = mass mixing ratio [1]\n"
01832             "# $8  = H2O volume mixing ratio [1]\n"
01833             "# $9  = O3 volume mixing ratio [1]\n"
01834             "# $10 = mean BT index [K]\n");
01835
01836   /* Set grid box size... */
01837   dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01838   dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01839   dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01840 }
01841
01842 /* Set time interval... */
01843 t0 = t - 0.5 * ctl->dt_mod;
01844 t1 = t + 0.5 * ctl->dt_mod;
01845
01846 /* Initialize... */
01847 for (ix = 0; ix < ctl->prof_nx; ix++)
01848   for (iy = 0; iy < ctl->prof_ny; iy++) {
01849     obsmean[ix][iy] = 0;
01850     obscount[ix][iy] = 0;
01851     tmean[ix][iy] = 0;
01852     for (iz = 0; iz < ctl->prof_nz; iz++)
01853       mass[ix][iy][iz] = 0;
01854   }
01855
01856 /* Read data... */
01857 while (fgets(line, LEN, in)) {
01858
01859   /* Read data... */
01860   if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01861     continue;
01862
01863   /* Check time... */
01864   if (rt < t0)
01865     continue;
01866   if (rt > t1)
01867     break;
01868
01869   /* Calculate indices... */
01870   ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01871   iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01872
01873   /* Check indices... */
01874   if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01875     continue;
01876
01877   /* Get mean observation index... */
01878   obsmean[ix][iy] += robs;
01879   tmean[ix][iy] += rt;
01880   obscount[ix][iy]++;
01881 }
01882
01883 /* Analyze model data... */
01884 for (ip = 0; ip < atm->np; ip++) {
01885
01886   /* Check time... */
01887   if (atm->time[ip] < t0 || atm->time[ip] > t1)
01888     continue;
01889
01890   /* Get indices... */
01891   ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01892   iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01893   iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01894
01895   /* Check indices... */
01896   if (ix < 0 || ix >= ctl->prof_nx ||
01897       iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01898     continue;
01899
01900   /* Get total mass in grid cell... */
01901   mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01902 }
01903
01904 /* Extract profiles... */
01905 for (ix = 0; ix < ctl->prof_nx; ix++)
01906   for (iy = 0; iy < ctl->prof_ny; iy++)
01907     if (obscount[ix][iy] > 0) {
01908
01909       /* Write output... */
01910       fprintf(out, "\n");
01911
```

```
01912         /* Loop over altitudes... */
01913         for (iz = 0; iz < ctl->prof_nz; iz++) {
01914
01915           /* Set coordinates... */
01916           z = ctl->prof_z0 + dz * (iz + 0.5);
01917           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01918           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01919
01920           /* Get meteorological data... */
01921           press = P(z);
01922           intpol_met_time(met0, met1, t, press, lon, lat,
01923                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01924
01925           /* Calculate mass mixing ratio... */
01926           rho_air = 100. * press / (287.058 * temp);
01927           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
01928             * cos(lat * M_PI / 180.);
01929           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01930
01931           /* Write output... */
01932           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01933                   tmean[ix][iy] / obscount[ix][iy],
01934                   z, lon, lat, press, temp, mmr, h2o, o3,
01935                   obsmean[ix][iy] / obscount[ix][iy]);
01936         }
01937       }
01938
01939   /* Close file... */
01940   if (t == ctl->t_stop)
01941     fclose(out);
01942 }
```

Here is the call graph for this function:



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

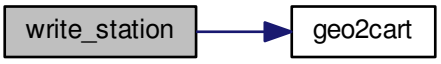
Write station data.

Definition at line 1946 of file libtrac.c.

```
01950             {
01951
01952   static FILE *out;
01953
01954   static double rmax2, t0, t1, x0[3], x1[3];
01955
01956   static int init, ip, iq;
01957
01958   /* Init... */
01959   if (!init) {
01960     init = 1;
01961
01962     /* Write info... */
01963     printf("Write station data: %s\n", filename);
01964
01965     /* Create new file... */
01966     if (!(out = fopen(filename, "w")))
01967       ERRMSG("Cannot create file!");
01968
01969     /* Write header... */
```

```
01970     fprintf(out,
01971             "# $1 = time [s]\n"
01972             "# $2 = altitude [km]\n"
01973             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01974     for (iq = 0; iq < ctl->nq; iq++)
01975       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01976               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01977     fprintf(out, "\n");
01978
01979     /* Set geolocation and search radius... */
01980     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01981     rmax2 = gsl_pow_2(ctl->stat_r);
01982   }
01983
01984   /* Set time interval for output... */
01985   t0 = t - 0.5 * ctl->dt_mod;
01986   t1 = t + 0.5 * ctl->dt_mod;
01987
01988   /* Loop over air parcels... */
01989   for (ip = 0; ip < atm->np; ip++) {
01990
01991     /* Check time... */
01992     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01993       continue;
01994
01995     /* Check station flag... */
01996     if (ctl->qnt_stat >= 0)
01997       if (atm->q[ctl->qnt_stat][ip])
01998         continue;
01999
02000     /* Get Cartesian coordinates... */
02001     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02002
02003     /* Check horizontal distance... */
02004     if (DIST2(x0, x1) > rmax2)
02005       continue;
02006
02007     /* Set station flag... */
02008     if (ctl->qnt_stat >= 0)
02009       atm->q[ctl->qnt_stat][ip] = 1;
02010
02011     /* Write data... */
02012     fprintf(out, "%.2f %g %g %g",
02013             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02014     for (iq = 0; iq < ctl->nq; iq++) {
02015       fprintf(out, " ");
02016       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02017     }
02018     fprintf(out, "\n");
02019   }
02020
02021   /* Close file... */
02022   if (t == ctl->t_stop)
02023     fclose(out);
02024 }
```

Here is the call graph for this function:



5.12 libtrac.c

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

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

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

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

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

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

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

```
00532       ctl->qnt_w = iq;
00533       sprintf(ctl->qnt_unit[iq], "hPa/s");
00534     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00535       ctl->qnt_h2o = iq;
00536       sprintf(ctl->qnt_unit[iq], "1");
00537     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00538       ctl->qnt_o3 = iq;
00539       sprintf(ctl->qnt_unit[iq], "1");
00540     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00541       ctl->qnt_theta = iq;
00542       sprintf(ctl->qnt_unit[iq], "K");
00543     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00544       ctl->qnt_pv = iq;
00545       sprintf(ctl->qnt_unit[iq], "PVU");
00546     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00547       ctl->qnt_tice = iq;
00548       sprintf(ctl->qnt_unit[iq], "K");
00549     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00550       ctl->qnt_tsts = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00553       ctl->qnt_tnat = iq;
00554       sprintf(ctl->qnt_unit[iq], "K");
00555     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00556       ctl->qnt_gw_var = iq;
00557       sprintf(ctl->qnt_unit[iq], "K^2");
00558     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00559       ctl->qnt_stat = iq;
00560       sprintf(ctl->qnt_unit[iq], "-");
00561     } else
00562       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00563   }
00564
00565   /* Time steps of simulation... */
00566   ctl->direction =
00567     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00568   if (ctl->direction != -1 && ctl->direction != 1)
00569     ERRMSG("Set DIRECTION to -1 or 1!");
00570   ctl->t_start =
00571     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00572   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00573   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00574
00575   /* Meteorological data... */
00576   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00577   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00578   if (ctl->met_np > EP)
00579     ERRMSG("Too many levels!");
00580   for (ip = 0; ip < ctl->met_np; ip++)
00581     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00582
00583   /* Isosurface parameters... */
00584   ctl->isosurf
00585     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00586   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00587
00588   /* Diffusion parameters... */
00589   ctl->turb_dx_trop
00590     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00591   ctl->turb_dx_strat
00592     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00593   ctl->turb_dz_trop
00594     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00595   ctl->turb_dz_strat
00596     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00597   ctl->turb_meso =
00598     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00599
00600   /* Life time of particles... */
00601   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00602   ctl->tdec_strat =
00603     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00604
00605   /* PSC analysis... */
00606   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00607   ctl->psc_hno3 =
00608     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00609
00610   /* Gravity wave analysis... */
00611   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
00612
00613   /* Output of atmospheric data... */
00614   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00615   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00616   ctl->atm_dt_out =
```

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

```
00700
00701 /*****************************************************************************/
00702
00703 void read_met(
00704   ctl_t * ctl,
00705   char *filename,
00706   met_t * met) {
00707
00708   char tstr[10];
00709
00710   static float help[EX * EY];
00711
00712   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00713
00714   size_t np, nx, ny;
00715
00716   /* Write info... */
00717   printf("Read meteorological data: %s\n", filename);
00718
00719   /* Get time from filename... */
00720   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00721   year = atoi(tstr);
00722   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00723   mon = atoi(tstr);
00724   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00725   day = atoi(tstr);
00726   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00727   hour = atoi(tstr);
00728   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00729
00730   /* Open netCDF file... */
00731   NC(nc_open(filename, NC_NOWRITE, &ncid));
00732
00733   /* Get dimensions... */
00734   NC(nc_inq_dimid(ncid, "lon", &dimid));
00735   NC(nc_inq_dimlen(ncid, dimid, &nx));
00736   if (nx > EX)
00737     ERRMSG("Too many longitudes!");
00738
00739   NC(nc_inq_dimid(ncid, "lat", &dimid));
00740   NC(nc_inq_dimlen(ncid, dimid, &ny));
00741   if (ny > EY)
00742     ERRMSG("Too many latitudes!");
00743
00744   NC(nc_inq_dimid(ncid, "lev", &dimid));
00745   NC(nc_inq_dimlen(ncid, dimid, &np));
00746   if (np > EP)
00747     ERRMSG("Too many levels!");
00748
00749   /* Store dimensions... */
00750   met->np = (int) np;
00751   met->nx = (int) nx;
00752   met->ny = (int) ny;
00753
00754   /* Get horizontal grid... */
00755   NC(nc_inq_varid(ncid, "lon", &varid));
00756   NC(nc_get_var_double(ncid, varid, met->lon));
00757   NC(nc_inq_varid(ncid, "lat", &varid));
00758   NC(nc_get_var_double(ncid, varid, met->lat));
00759
00760   /* Read meteorological data... */
00761   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00762   read_met_help(ncid, "u", "U", met, met->u, 1.0);
00763   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00764   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00765   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00766   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00767
00768   /* Meteo data on pressure levels... */
00769   if (ctl->met_np <= 0) {
00770
00771     /* Read pressure levels from file... */
00772     NC(nc_inq_varid(ncid, "lev", &varid));
00773     NC(nc_get_var_double(ncid, varid, met->p));
00774     for (ip = 0; ip < met->np; ip++)
00775       met->p[ip] /= 100.;
00776
00777     /* Extrapolate data for lower boundary... */
00778     read_met_extrapolate(met);
00779   }
00780
00781   /* Meteo data on model levels... */
00782   else {
00783
00784     /* Read pressure data from file... */
00785     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00786
```

```
00787     /* Interpolate from model levels to pressure levels... */
00788     read_met_ml2pl(ctl, met, met->t);
00789     read_met_ml2pl(ctl, met, met->u);
00790     read_met_ml2pl(ctl, met, met->v);
00791     read_met_ml2pl(ctl, met, met->w);
00792     read_met_ml2pl(ctl, met, met->h2o);
00793     read_met_ml2pl(ctl, met, met->o3);
00794
00795     /* Set pressure levels... */
00796     met->np = ctl->met_np;
00797     for (ip = 0; ip < met->np; ip++)
00798       met->p[ip] = ctl->met_p[ip];
00799   }
00800
00801   /* Check ordering of pressure levels... */
00802   for (ip = 1; ip < met->np; ip++)
00803     if (met->p[ip - 1] < met->p[ip])
00804       ERRMSG("Pressure levels must be descending!");
00805
00806   /* Read surface pressure... */
00807   if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00808     NC(nc_get_var_float(ncid, varid, help));
00809     for (iy = 0; iy < met->ny; iy++)
00810       for (ix = 0; ix < met->nx; ix++)
00811         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00812   } else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00813     NC(nc_get_var_float(ncid, varid, help));
00814     for (iy = 0; iy < met->ny; iy++)
00815       for (ix = 0; ix < met->nx; ix++)
00816         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00817   } else
00818     for (ix = 0; ix < met->nx; ix++)
00819       for (iy = 0; iy < met->ny; iy++)
00820         met->ps[ix][iy] = met->p[0];
00821
00822   /* Create periodic boundary conditions... */
00823   read_met_periodic(met);
00824
00825   /* Close file... */
00826   NC(nc_close(ncid));
00827 }
00828
00829 /*****************************************************************************/
00830
00831 void read_met_extrapolate(
00832   met_t * met) {
00833
00834   int ip, ip0, ix, iy;
00835
00836   /* Loop over columns... */
00837   for (ix = 0; ix < met->nx; ix++)
00838     for (iy = 0; iy < met->ny; iy++) {
00839
00840       /* Find lowest valid data point... */
00841       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00842         if (!gsl_finite(met->t[ix][iy][ip0])
00843             || !gsl_finite(met->u[ix][iy][ip0])
00844             || !gsl_finite(met->v[ix][iy][ip0])
00845             || !gsl_finite(met->w[ix][iy][ip0]))
00846           break;
00847
00848       /* Extrapolate... */
00849       for (ip = ip0; ip >= 0; ip--) {
00850         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00851         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00852         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00853         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00854         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00855         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00856       }
00857     }
00858 }
00859
00860 /*****************************************************************************/
00861
00862 void read_met_help(
00863   int ncid,
00864   char *varname,
00865   char *varname2,
00866   met_t * met,
00867   float dest[EX][EY][EP],
00868   float scl) {
00869
00870   static float help[EX * EY * EP];
00871
00872   int ip, ix, iy, n = 0, varid;
00873
```

```
00874   /* Check if variable exists... */
00875   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00876     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00877       return;
00878
00879   /* Read data... */
00880   NC(nc_get_var_float(ncid, varid, help));
00881
00882   /* Copy and check data... */
00883   for (ip = 0; ip < met->np; ip++)
00884     for (iy = 0; iy < met->ny; iy++)
00885       for (ix = 0; ix < met->nx; ix++) {
00886         dest[ix][iy][ip] = scl * help[n++];
00887         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00888           dest[ix][iy][ip] = GSL_NAN;
00889       }
00890 }
00891
00892 /*****************************************************************************/
00893
00894 void read_met_ml2pl(
00895   ctl_t * ctl,
00896   met_t * met,
00897   float var[EX][EY][EP]) {
00898
00899   double aux[EP], p[EP], pt;
00900
00901   int ip, ip2, ix, iy;
00902
00903   /* Loop over columns... */
00904   for (ix = 0; ix < met->nx; ix++)
00905     for (iy = 0; iy < met->ny; iy++) {
00906
00907       /* Copy pressure profile... */
00908       for (ip = 0; ip < met->np; ip++)
00909         p[ip] = met->pl[ix][iy][ip];
00910
00911       /* Interpolate... */
00912       for (ip = 0; ip < ctl->met_np; ip++) {
00913         pt = ctl->met_p[ip];
00914         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00915           pt = p[0];
00916         else if ((pt > p[met->np - 1] && p[1] > p[0])
00917                  || (pt < p[met->np - 1] && p[1] < p[0]))
00918           pt = p[met->np - 1];
00919         ip2 = locate(p, met->np, pt);
00920         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00921                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00922       }
00923
00924       /* Copy data... */
00925       for (ip = 0; ip < ctl->met_np; ip++)
00926         var[ix][iy][ip] = (float) aux[ip];
00927     }
00928 }
00929
00930 /*****************************************************************************/
00931
00932 void read_met_periodic(
00933   met_t * met) {
00934
00935   int ip, iy;
00936
00937   /* Check longitudes... */
00938   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00939              + met->lon[1] - met->lon[0] - 360) < 0.01))
00940     return;
00941
00942   /* Increase longitude counter... */
00943   if ((++met->nx) > EX)
00944     ERRMSG("Cannot create periodic boundary conditions!");
00945
00946   /* Set longitude... */
00947   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00948
00949   /* Loop over latitudes and pressure levels... */
00950   for (iy = 0; iy < met->ny; iy++)
00951     for (ip = 0; ip < met->np; ip++) {
00952       met->ps[met->nx - 1][iy] = met->ps[0][iy];
00953       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
00954       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00955       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00956       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00957       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00958       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00959     }
```

```
00960 }
00961
00962 /*****************************************************************************/
00963
00964 double scan_ctl(
00965   const char *filename,
00966   int argc,
00967   char *argv[],
00968   const char *varname,
00969   int arridx,
00970   const char *defvalue,
00971   char *value) {
00972
00973   FILE *in = NULL;
00974
00975   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00976     msg[LEN], rvarname[LEN], rval[LEN];
00977
00978   int contain = 0, i;
00979
00980   /* Open file... */
00981   if (filename[strlen(filename) - 1] != '-')
00982     if (!(in = fopen(filename, "r")))
00983       ERRMSG("Cannot open file!");
00984
00985   /* Set full variable name... */
00986   if (arridx >= 0) {
00987     sprintf(fullname1, "%s[%d]", varname, arridx);
00988     sprintf(fullname2, "%s[*]", varname);
00989   } else {
00990     sprintf(fullname1, "%s", varname);
00991     sprintf(fullname2, "%s", varname);
00992   }
00993
00994   /* Read data... */
00995   if (in != NULL)
00996     while (fgets(line, LEN, in))
00997       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00998         if (strcasecmp(rvarname, fullname1) == 0 ||
00999             strcasecmp(rvarname, fullname2) == 0) {
01000           contain = 1;
01001           break;
01002         }
01003   for (i = 1; i < argc - 1; i++)
01004     if (strcasecmp(argv[i], fullname1) == 0 ||
01005         strcasecmp(argv[i], fullname2) == 0) {
01006       sprintf(rval, "%s", argv[i + 1]);
01007       contain = 1;
01008       break;
01009     }
01010
01011   /* Close file... */
01012   if (in != NULL)
01013     fclose(in);
01014
01015   /* Check for missing variables... */
01016   if (!contain) {
01017     if (strlen(defvalue) > 0)
01018       sprintf(rval, "%s", defvalue);
01019     else {
01020       sprintf(msg, "Missing variable %s!\n", fullname1);
01021       ERRMSG(msg);
01022     }
01023   }
01024
01025   /* Write info... */
01026   printf("%s = %s\n", fullname1, rval);
01027
01028   /* Return values... */
01029   if (value != NULL)
01030     sprintf(value, "%s", rval);
01031   return atof(rval);
01032 }
01033
01034 /*****************************************************************************/
01035
01036 void time2jsec(
01037   int year,
01038   int mon,
01039   int day,
01040   int hour,
01041   int min,
01042   int sec,
01043   double remain,
01044   double *jsec) {
01045
01046   struct tm t0, t1;
```

```
01047
01048   t0.tm_year = 100;
01049   t0.tm_mon = 0;
01050   t0.tm_mday = 1;
01051   t0.tm_hour = 0;
01052   t0.tm_min = 0;
01053   t0.tm_sec = 0;
01054
01055   t1.tm_year = year - 1900;
01056   t1.tm_mon = mon - 1;
01057   t1.tm_mday = day;
01058   t1.tm_hour = hour;
01059   t1.tm_min = min;
01060   t1.tm_sec = sec;
01061
01062   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01063 }
01064
01065 /*****************************************************************************/
01066
01067 void timer(
01068   const char *name,
01069   int id,
01070   int mode) {
01071
01072   static double starttime[NTIMER], runtime[NTIMER];
01073
01074   /* Check id... */
01075   if (id < 0 || id >= NTIMER)
01076     ERRMSG("Too many timers!");
01077
01078   /* Start timer... */
01079   if (mode == 1) {
01080     if (starttime[id] <= 0)
01081       starttime[id] = omp_get_wtime();
01082     else
01083       ERRMSG("Timer already started!");
01084   }
01085
01086   /* Stop timer... */
01087   else if (mode == 2) {
01088     if (starttime[id] > 0) {
01089       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01090       starttime[id] = -1;
01091     } else
01092       ERRMSG("Timer not started!");
01093   }
01094
01095   /* Print timer... */
01096   else if (mode == 3)
01097     printf("%s = %g s\n", name, runtime[id]);
01098 }
01099
01100 /*****************************************************************************/
01101
01102 double tropopause(
01103   double t,
01104   double lat) {
01105
01106   static double doys[12]
01107   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01108
01109   static double lats[73]
01110     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01111     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01112     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01113     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01114     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01115     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01116     75, 77.5, 80, 82.5, 85, 87.5, 90
01117   };
01118
01119   static double tps[12][73]
01120     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01121          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01122          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01123          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01124          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01125          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01126          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01127          275.3, 275.6, 275.4, 274.1, 273.5},
01128   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01129    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01130    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01131    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01132    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01133    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
```

```
01134    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01135    287.5, 286.2, 285.8},
01136   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01137    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01138    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01139    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01140    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01141    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01142    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01143    304.3, 304.9, 306, 306.6, 306.2, 306},
01144   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01145    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01146    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01147    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01148    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01149    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01150    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01151    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01152   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01153    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01154    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01155    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01156    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01157    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01158    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01159    325.3, 325.8, 325.8},
01160   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01161    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01162    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01163    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01164    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01165    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01166    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01167    308.5, 312.2, 313.1, 313.3},
01168   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01169    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01170    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01171    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01172    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01173    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01174    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01175    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01176   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01177    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01178    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01179    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01180    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01181    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01182    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01183    278.2, 282.6, 287.4, 290.9, 292.5, 293},
01184   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01185    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01186    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01187    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01188    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01189    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01190    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01191    276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01192   {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01193    215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01194    237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01195    111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01196    106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01197    112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01198    206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01199    279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01200    305.1},
01201   {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01202    253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01203    223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01204    108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01205    102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01206    109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01207    241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01208    286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01209   {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01210    284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01211    175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01212    100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01213    100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01214    186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01215    280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01216    281.7, 281.1, 281.2}
01217   };
01218
01219   double doy, p0, p1, pt;
01220
```

```
01221   int imon, ilat;
01222
01223   /* Get day of year... */
01224   doy = fmod(t / 86400., 365.25);
01225   while (doy < 0)
01226     doy += 365.25;
01227
01228   /* Get indices... */
01229   imon = locate(doys, 12, doy);
01230   ilat = locate(lats, 73, lat);
01231
01232   /* Get tropopause pressure... */
01233   p0 = LIN(lats[ilat], tps[imon][ilat],
01234            lats[ilat + 1], tps[imon][ilat + 1], lat);
01235   p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01236            lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01237   pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01238
01239   /* Return tropopause pressure... */
01240   return pt;
01241 }
01242
01243 /*****************************************************************************/
01244
01245 void write_atm(
01246   const char *filename,
01247   ctl_t * ctl,
01248   atm_t * atm,
01249   double t) {
01250
01251   FILE *in, *out;
01252
01253   char line[LEN];
01254
01255   double r, t0, t1;
01256
01257   int ip, iq, year, mon, day, hour, min, sec;
01258
01259   /* Set time interval for output... */
01260   t0 = t - 0.5 * ctl->dt_mod;
01261   t1 = t + 0.5 * ctl->dt_mod;
01262
01263   /* Check if gnuplot output is requested... */
01264   if (ctl->atm_gpfile[0] != '-') {
01265
01266     /* Write info... */
01267     printf("Plot atmospheric data: %s.png\n", filename);
01268
01269     /* Create gnuplot pipe... */
01270     if (!(out = popen("gnuplot", "w")))
01271       ERRMSG("Cannot create pipe to gnuplot!");
01272
01273     /* Set plot filename... */
01274     fprintf(out, "set out \"%s.png\"\n", filename);
01275
01276     /* Set time string... */
01277     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01278     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01279             year, mon, day, hour, min);
01280
01281     /* Dump gnuplot file to pipe... */
01282     if (!(in = fopen(ctl->atm_gpfile, "r")))
01283       ERRMSG("Cannot open file!");
01284     while (fgets(line, LEN, in))
01285       fprintf(out, "%s", line);
01286     fclose(in);
01287   }
01288
01289   else {
01290
01291     /* Write info... */
01292     printf("Write atmospheric data: %s\n", filename);
01293
01294     /* Create file... */
01295     if (!(out = fopen(filename, "w")))
01296       ERRMSG("Cannot create file!");
01297   }
01298
01299   /* Write header... */
01300   fprintf(out,
01301           "# $1 = time [s]\n"
01302           "# $2 = altitude [km]\n"
01303           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01304   for (iq = 0; iq < ctl->nq; iq++)
01305     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01306             ctl->qnt_unit[iq]);
01307   fprintf(out, "\n");
```

```
01308
01309   /* Write data... */
01310   for (ip = 0; ip < atm->np; ip++) {
01311
01312     /* Check time... */
01313     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01314       continue;
01315
01316     /* Write output... */
01317     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01318             atm->lon[ip], atm->lat[ip]);
01319     for (iq = 0; iq < ctl->nq; iq++) {
01320       fprintf(out, " ");
01321       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01322     }
01323     fprintf(out, "\n");
01324   }
01325
01326   /* Close file... */
01327   fclose(out);
01328 }
01329
01330 /*****************************************************************************/
01331
01332 void write_csi(
01333   const char *filename,
01334   ctl_t * ctl,
01335   atm_t * atm,
01336   double t) {
01337
01338   static FILE *in, *out;
01339
01340   static char line[LEN];
01341
01342   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01343     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01344
01345   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01346
01347   /* Init... */
01348   if (!init) {
01349     init = 1;
01350
01351     /* Check quantity index for mass... */
01352     if (ctl->qnt_m < 0)
01353       ERRMSG("Need quantity mass to analyze CSI!");
01354
01355     /* Open observation data file... */
01356     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01357     if (!(in = fopen(ctl->csi_obsfile, "r")))
01358       ERRMSG("Cannot open file!");
01359
01360     /* Create new file... */
01361     printf("Write CSI data: %s\n", filename);
01362     if (!(out = fopen(filename, "w")))
01363       ERRMSG("Cannot create file!");
01364
01365     /* Write header... */
01366     fprintf(out,
01367             "# $1 = time [s]\n"
01368             "# $2 = number of hits (cx)\n"
01369             "# $3 = number of misses (cy)\n"
01370             "# $4 = number of false alarms (cz)\n"
01371             "# $5 = number of observations (cx + cy)\n"
01372             "# $6 = number of forecasts (cx + cz)\n"
01373             "# $7 = bias (forecasts/observations) [%%]\n"
01374             "# $8 = probability of detection (POD) [%%]\n"
01375             "# $9 = false alarm rate (FAR) [%%]\n"
01376             "# $10 = critical success index (CSI) [%%]\n\n");
01377   }
01378
01379   /* Set time interval... */
01380   t0 = t - 0.5 * ctl->dt_mod;
01381   t1 = t + 0.5 * ctl->dt_mod;
01382
01383   /* Initialize grid cells... */
01384   for (ix = 0; ix < ctl->csi_nx; ix++)
01385     for (iy = 0; iy < ctl->csi_ny; iy++)
01386       for (iz = 0; iz < ctl->csi_nz; iz++)
01387         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01388
01389   /* Read data... */
01390   while (fgets(line, LEN, in)) {
01391
01392     /* Read data... */
01393     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01394         5)
```

```
01395       continue;
01396
01397     /* Check time... */
01398     if (rt < t0)
01399       continue;
01400     if (rt > t1)
01401       break;
01402
01403     /* Calculate indices... */
01404     ix = (int) ((rlon - ctl->csi_lon0)
01405                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01406     iy = (int) ((rlat - ctl->csi_lat0)
01407                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01408     iz = (int) ((rz - ctl->csi_z0)
01409                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01410
01411     /* Check indices... */
01412     if (ix < 0 || ix >= ctl->csi_nx ||
01413         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01414       continue;
01415
01416     /* Get mean observation index... */
01417     obsmean[ix][iy][iz] += robs;
01418     obscount[ix][iy][iz]++;
01419   }
01420
01421   /* Analyze model data... */
01422   for (ip = 0; ip < atm->np; ip++) {
01423
01424     /* Check time... */
01425     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01426       continue;
01427
01428     /* Get indices... */
01429     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01430                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01431     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01432                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01433     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01434                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01435
01436     /* Check indices... */
01437     if (ix < 0 || ix >= ctl->csi_nx ||
01438         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01439       continue;
01440
01441     /* Get total mass in grid cell... */
01442     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01443   }
01444
01445   /* Analyze all grid cells... */
01446   for (ix = 0; ix < ctl->csi_nx; ix++)
01447     for (iy = 0; iy < ctl->csi_ny; iy++)
01448       for (iz = 0; iz < ctl->csi_nz; iz++) {
01449
01450         /* Calculate mean observation index... */
01451         if (obscount[ix][iy][iz] > 0)
01452           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01453
01454         /* Calculate column density... */
01455         if (modmean[ix][iy][iz] > 0) {
01456           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01457           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01458           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01459           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01460             * cos(lat * M_PI / 180.);
01461           modmean[ix][iy][iz] /= (1e6 * area);
01462         }
01463
01464         /* Calculate CSI... */
01465         if (obscount[ix][iy][iz] > 0) {
01466           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01467               modmean[ix][iy][iz] >= ctl->csi_modmin)
01468             cx++;
01469           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01470                    modmean[ix][iy][iz] < ctl->csi_modmin)
01471             cy++;
01472           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01473                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01474             cz++;
01475         }
01476       }
01477
01478   /* Write output... */
01479   if (fmod(t, ctl->csi_dt_out) == 0) {
01480
01481     /* Write... */
```

```
01482     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01483             t, cx, cy, cz, cx + cy, cx + cz,
01484             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01485             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01486             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01487             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01488
01489     /* Set counters to zero... */
01490     cx = cy = cz = 0;
01491   }
01492
01493   /* Close file... */
01494   if (t == ctl->t_stop)
01495     fclose(out);
01496 }
01497
01498 /*****************************************************************************/
01499
01500 void write_ens(
01501   const char *filename,
01502   ctl_t * ctl,
01503   atm_t * atm,
01504   double t) {
01505
01506   static FILE *out;
01507
01508   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01509     t0, t1, x[NENS][3], xm[3];
01510
01511   static int init, ip, iq;
01512
01513   static size_t i, n;
01514
01515   /* Init... */
01516   if (!init) {
01517     init = 1;
01518
01519     /* Check quantities... */
01520     if (ctl->qnt_ens < 0)
01521       ERRMSG("Missing ensemble IDs!");
01522
01523     /* Create new file... */
01524     printf("Write ensemble data: %s\n", filename);
01525     if (!(out = fopen(filename, "w")))
01526       ERRMSG("Cannot create file!");
01527
01528     /* Write header... */
01529     fprintf(out,
01530             "# $1 = time [s]\n"
01531             "# $2 = altitude [km]\n"
01532             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01533     for (iq = 0; iq < ctl->nq; iq++)
01534       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01535               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01536     for (iq = 0; iq < ctl->nq; iq++)
01537       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01538               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01539     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01540   }
01541
01542   /* Set time interval... */
01543   t0 = t - 0.5 * ctl->dt_mod;
01544   t1 = t + 0.5 * ctl->dt_mod;
01545
01546   /* Init... */
01547   ens = GSL_NAN;
01548   n = 0;
01549
01550   /* Loop over air parcels... */
01551   for (ip = 0; ip < atm->np; ip++) {
01552
01553     /* Check time... */
01554     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01555       continue;
01556
01557     /* Check ensemble id... */
01558     if (atm->q[ctl->qnt_ens][ip] != ens) {
01559
01560       /* Write results... */
01561       if (n > 0) {
01562
01563         /* Get mean position... */
01564         xm[0] = xm[1] = xm[2] = 0;
01565         for (i = 0; i < n; i++) {
01566           xm[0] += x[i][0] / (double) n;
01567           xm[1] += x[i][1] / (double) n;
01568           xm[2] += x[i][2] / (double) n;
```

```
01569         }
01570         cart2geo(xm, &dummy, &lon, &lat);
01571         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01572                 lat);
01573
01574         /* Get quantity statistics... */
01575         for (iq = 0; iq < ctl->nq; iq++) {
01576           fprintf(out, " ");
01577           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01578         }
01579         for (iq = 0; iq < ctl->nq; iq++) {
01580           fprintf(out, " ");
01581           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01582         }
01583         fprintf(out, " %lu\n", n);
01584       }
01585
01586       /* Init new ensemble... */
01587       ens = atm->q[ctl->qnt_ens][ip];
01588       n = 0;
01589     }
01590
01591     /* Save data... */
01592     p[n] = atm->p[ip];
01593     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01594     for (iq = 0; iq < ctl->nq; iq++)
01595       q[iq][n] = atm->q[iq][ip];
01596     if ((++n) >= NENS)
01597       ERRMSG("Too many data points!");
01598   }
01599
01600   /* Write results... */
01601   if (n > 0) {
01602
01603     /* Get mean position... */
01604     xm[0] = xm[1] = xm[2] = 0;
01605     for (i = 0; i < n; i++) {
01606       xm[0] += x[i][0] / (double) n;
01607       xm[1] += x[i][1] / (double) n;
01608       xm[2] += x[i][2] / (double) n;
01609     }
01610     cart2geo(xm, &dummy, &lon, &lat);
01611     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01612
01613     /* Get quantity statistics... */
01614     for (iq = 0; iq < ctl->nq; iq++) {
01615       fprintf(out, " ");
01616       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01617     }
01618     for (iq = 0; iq < ctl->nq; iq++) {
01619       fprintf(out, " ");
01620       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01621     }
01622     fprintf(out, " %lu\n", n);
01623   }
01624
01625   /* Close file... */
01626   if (t == ctl->t_stop)
01627     fclose(out);
01628 }
01629
01630 /*****************************************************************************/
01631
01632 void write_grid(
01633   const char *filename,
01634   ctl_t * ctl,
01635   met_t * met0,
01636   met_t * met1,
01637   atm_t * atm,
01638   double t) {
01639
01640   FILE *in, *out;
01641
01642   char line[LEN];
01643
01644   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01645     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01646
01647   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01648
01649   /* Check dimensions... */
01650   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01651     ERRMSG("Grid dimensions too large!");
01652
01653   /* Check quantity index for mass... */
01654   if (ctl->qnt_m < 0)
01655     ERRMSG("Need quantity mass to write grid data!");
```

```
01656
01657   /* Set time interval for output... */
01658   t0 = t - 0.5 * ctl->dt_mod;
01659   t1 = t + 0.5 * ctl->dt_mod;
01660
01661   /* Set grid box size... */
01662   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01663   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01664   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01665
01666   /* Initialize grid... */
01667   for (ix = 0; ix < ctl->grid_nx; ix++)
01668     for (iy = 0; iy < ctl->grid_ny; iy++)
01669       for (iz = 0; iz < ctl->grid_nz; iz++)
01670         grid_m[ix][iy][iz] = 0;
01671
01672   /* Average data... */
01673   for (ip = 0; ip < atm->np; ip++)
01674     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01675
01676       /* Get index... */
01677       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01678       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01679       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01680
01681       /* Check indices... */
01682       if (ix < 0 || ix >= ctl->grid_nx ||
01683           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01684         continue;
01685
01686       /* Add mass... */
01687       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01688     }
01689
01690   /* Check if gnuplot output is requested... */
01691   if (ctl->grid_gpfile[0] != '-') {
01692
01693     /* Write info... */
01694     printf("Plot grid data: %s.png\n", filename);
01695
01696     /* Create gnuplot pipe... */
01697     if (!(out = popen("gnuplot", "w")))
01698       ERRMSG("Cannot create pipe to gnuplot!");
01699
01700     /* Set plot filename... */
01701     fprintf(out, "set out \"%s.png\"\n", filename);
01702
01703     /* Set time string... */
01704     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01705     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01706             year, mon, day, hour, min);
01707
01708     /* Dump gnuplot file to pipe... */
01709     if (!(in = fopen(ctl->grid_gpfile, "r")))
01710       ERRMSG("Cannot open file!");
01711     while (fgets(line, LEN, in))
01712       fprintf(out, "%s", line);
01713     fclose(in);
01714   }
01715
01716   else {
01717
01718     /* Write info... */
01719     printf("Write grid data: %s\n", filename);
01720
01721     /* Create file... */
01722     if (!(out = fopen(filename, "w")))
01723       ERRMSG("Cannot create file!");
01724   }
01725
01726   /* Write header... */
01727   fprintf(out,
01728           "# $1 = time [s]\n"
01729           "# $2 = altitude [km]\n"
01730           "# $3 = longitude [deg]\n"
01731           "# $4 = latitude [deg]\n"
01732           "# $5 = surface area [km^2]\n"
01733           "# $6 = layer width [km]\n"
01734           "# $7 = temperature [K]\n"
01735           "# $8 = column density [kg/m^2]\n"
01736           "# $9 = mass mixing ratio [1]\n\n");
01737
01738   /* Write data... */
01739   for (ix = 0; ix < ctl->grid_nx; ix++) {
01740     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01741       fprintf(out, "\n");
01742     for (iy = 0; iy < ctl->grid_ny; iy++) {
```

```
01743       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01744         fprintf(out, "\n");
01745       for (iz = 0; iz < ctl->grid_nz; iz++)
01746         if (!ctl->grid_sparse
01747             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01748
01749           /* Set coordinates... */
01750           z = ctl->grid_z0 + dz * (iz + 0.5);
01751           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01752           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01753
01754           /* Get pressure and temperature... */
01755           press = P(z);
01756           intpol_met_time(met0, met1, t, press, lon, lat,
01757                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01758
01759           /* Calculate surface area... */
01760           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01761             * cos(lat * M_PI / 180.);
01762
01763           /* Calculate column density... */
01764           cd = grid_m[ix][iy][iz] / (1e6 * area);
01765
01766           /* Calculate mass mixing ratio... */
01767           rho_air = 100. * press / (287.058 * temp);
01768           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01769
01770           /* Write output... */
01771           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01772                   t, z, lon, lat, area, dz, temp, cd, mmr);
01773         }
01774     }
01775   }
01776
01777   /* Close file... */
01778   fclose(out);
01779 }
01780
01781 /*****************************************************************************/
01782
01783 void write_prof(
01784   const char *filename,
01785   ctl_t * ctl,
01786   met_t * met0,
01787   met_t * met1,
01788   atm_t * atm,
01789   double t) {
01790
01791   static FILE *in, *out;
01792
01793   static char line[LEN];
01794
01795   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01796     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01797     press, temp, rho_air, mmr, h2o, o3;
01798
01799   static int init, obscount[GX][GY], ip, ix, iy, iz;
01800
01801   /* Init... */
01802   if (!init) {
01803     init = 1;
01804
01805     /* Check quantity index for mass... */
01806     if (ctl->qnt_m < 0)
01807       ERRMSG("Need quantity mass!");
01808
01809     /* Check dimensions... */
01810     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01811       ERRMSG("Grid dimensions too large!");
01812
01813     /* Open observation data file... */
01814     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01815     if (!(in = fopen(ctl->prof_obsfile, "r")))
01816       ERRMSG("Cannot open file!");
01817
01818     /* Create new file... */
01819     printf("Write profile data: %s\n", filename);
01820     if (!(out = fopen(filename, "w")))
01821       ERRMSG("Cannot create file!");
01822
01823     /* Write header... */
01824     fprintf(out,
01825             "# $1  = time [s]\n"
01826             "# $2  = altitude [km]\n"
01827             "# $3  = longitude [deg]\n"
01828             "# $4  = latitude [deg]\n"
01829             "# $5  = pressure [hPa]\n"
```

```
01830             "# $6  = temperature [K]\n"
01831             "# $7  = mass mixing ratio [1]\n"
01832             "# $8  = H2O volume mixing ratio [1]\n"
01833             "# $9  = O3 volume mixing ratio [1]\n"
01834             "# $10 = mean BT index [K]\n");
01835
01836     /* Set grid box size... */
01837     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01838     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01839     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01840   }
01841
01842   /* Set time interval... */
01843   t0 = t - 0.5 * ctl->dt_mod;
01844   t1 = t + 0.5 * ctl->dt_mod;
01845
01846   /* Initialize... */
01847   for (ix = 0; ix < ctl->prof_nx; ix++)
01848     for (iy = 0; iy < ctl->prof_ny; iy++) {
01849       obsmean[ix][iy] = 0;
01850       obscount[ix][iy] = 0;
01851       tmean[ix][iy] = 0;
01852       for (iz = 0; iz < ctl->prof_nz; iz++)
01853         mass[ix][iy][iz] = 0;
01854     }
01855
01856   /* Read data... */
01857   while (fgets(line, LEN, in)) {
01858
01859     /* Read data... */
01860     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01861       continue;
01862
01863     /* Check time... */
01864     if (rt < t0)
01865       continue;
01866     if (rt > t1)
01867       break;
01868
01869     /* Calculate indices... */
01870     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01871     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01872
01873     /* Check indices... */
01874     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01875       continue;
01876
01877     /* Get mean observation index... */
01878     obsmean[ix][iy] += robs;
01879     tmean[ix][iy] += rt;
01880     obscount[ix][iy]++;
01881   }
01882
01883   /* Analyze model data... */
01884   for (ip = 0; ip < atm->np; ip++) {
01885
01886     /* Check time... */
01887     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01888       continue;
01889
01890     /* Get indices... */
01891     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01892     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01893     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01894
01895     /* Check indices... */
01896     if (ix < 0 || ix >= ctl->prof_nx ||
01897         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01898       continue;
01899
01900     /* Get total mass in grid cell... */
01901     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01902   }
01903
01904   /* Extract profiles... */
01905   for (ix = 0; ix < ctl->prof_nx; ix++)
01906     for (iy = 0; iy < ctl->prof_ny; iy++)
01907       if (obscount[ix][iy] > 0) {
01908
01909         /* Write output... */
01910         fprintf(out, "\n");
01911
01912         /* Loop over altitudes... */
01913         for (iz = 0; iz < ctl->prof_nz; iz++) {
01914
01915           /* Set coordinates... */
01916           z = ctl->prof_z0 + dz * (iz + 0.5);
```

```
01917           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01918           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01919
01920           /* Get meteorological data... */
01921           press = P(z);
01922           intpol_met_time(met0, met1, t, press, lon, lat,
01923                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01924
01925           /* Calculate mass mixing ratio... */
01926           rho_air = 100. * press / (287.058 * temp);
01927           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
01928             * cos(lat * M_PI / 180.);
01929           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01930
01931           /* Write output... */
01932           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01933                   tmean[ix][iy] / obscount[ix][iy],
01934                   z, lon, lat, press, temp, mmr, h2o, o3,
01935                   obsmean[ix][iy] / obscount[ix][iy]);
01936         }
01937       }
01938
01939   /* Close file... */
01940   if (t == ctl->t_stop)
01941     fclose(out);
01942 }
01943
01944 /*****************************************************************************/
01945
01946 void write_station(
01947   const char *filename,
01948   ctl_t * ctl,
01949   atm_t * atm,
01950   double t) {
01951
01952   static FILE *out;
01953
01954   static double rmax2, t0, t1, x0[3], x1[3];
01955
01956   static int init, ip, iq;
01957
01958   /* Init... */
01959   if (!init) {
01960     init = 1;
01961
01962     /* Write info... */
01963     printf("Write station data: %s\n", filename);
01964
01965     /* Create new file... */
01966     if (!(out = fopen(filename, "w")))
01967       ERRMSG("Cannot create file!");
01968
01969     /* Write header... */
01970     fprintf(out,
01971             "# $1 = time [s]\n"
01972             "# $2 = altitude [km]\n"
01973             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01974     for (iq = 0; iq < ctl->nq; iq++)
01975       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01976               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01977     fprintf(out, "\n");
01978
01979     /* Set geolocation and search radius... */
01980     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01981     rmax2 = gsl_pow_2(ctl->stat_r);
01982   }
01983
01984   /* Set time interval for output... */
01985   t0 = t - 0.5 * ctl->dt_mod;
01986   t1 = t + 0.5 * ctl->dt_mod;
01987
01988   /* Loop over air parcels... */
01989   for (ip = 0; ip < atm->np; ip++) {
01990
01991     /* Check time... */
01992     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01993       continue;
01994
01995     /* Check station flag... */
01996     if (ctl->qnt_stat >= 0)
01997       if (atm->q[ctl->qnt_stat][ip])
01998         continue;
01999
02000     /* Get Cartesian coordinates... */
02001     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02002
02003     /* Check horizontal distance... */
```

```
02004     if (DIST2(x0, x1) > rmax2)
02005       continue;
02006
02007     /* Set station flag... */
02008     if (ctl->qnt_stat >= 0)
02009       atm->q[ctl->qnt_stat][ip] = 1;
02010
02011     /* Write data... */
02012     fprintf(out, "%.2f %g %g %g",
02013             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02014     for (iq = 0; iq < ctl->nq; iq++) {
02015       fprintf(out, " ");
02016       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02017     }
02018     fprintf(out, "\n");
02019   }
02020
02021   /* Close file... */
02022   if (t == ctl->t_stop)
02023     fclose(out);
02024 }
```

5.13 libtrac.h File Reference

MPTRAC library declarations.

Data Structures

- struct ctl_t

  Control parameters.
- struct atm_t

  Atmospheric data.
- struct met_t

  Meteorological data.

Functions

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

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

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

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

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

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

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

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

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

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

  Get meteorological data for timestep.

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

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

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

  Spatial interpolation of meteorological data.
- void intpol_met_time (met_t *met0, met_t *met1, double ts, double p, double lon, double lat, double *ps, double *t, double *u, double *v, double *w, double *h2o, double *o3)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Write station data.

5.13.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

5.13.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

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

5.13.2.2 double deg2dx (double dlon, double lat)

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

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

5.13.2.3 double deg2dy (double dlat)

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

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

5.13.2.4 double dp2dz (double dp, double p)

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

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

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

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

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

5.13.2.6 double dy2deg (double *dy*)

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

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

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

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

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

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

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

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

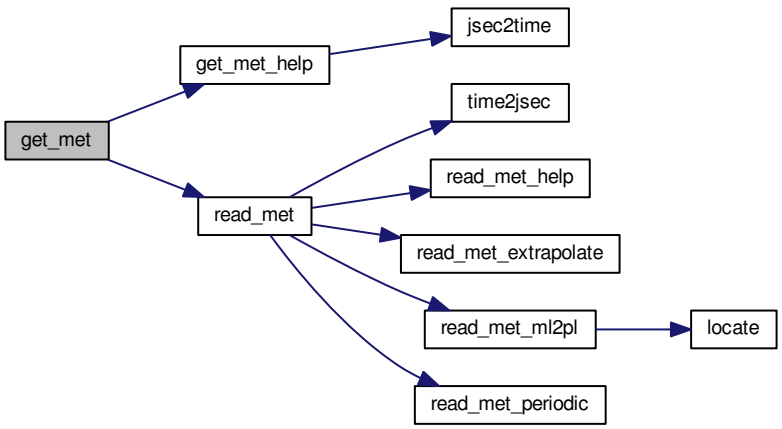
5.13.2.9 void get_met (ctl_t * *ctl,* char * *metbase,* double *t,* met_t * *met0,* met_t * *met1*)

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

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

Here is the call graph for this function:



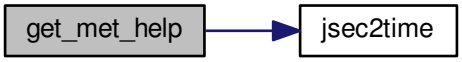
5.13.2.10 void get_met_help (double *t,* int *direct,* char * *metbase,* double *dt_met,* char * *filename*)

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

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

Here is the call graph for this function:



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

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

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

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

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

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

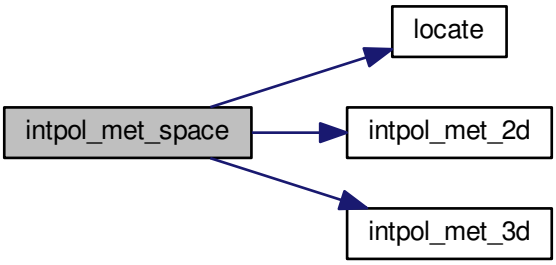
5.13.2.13 void intpol_met_space (met_t * *met*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

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

Here is the call graph for this function:



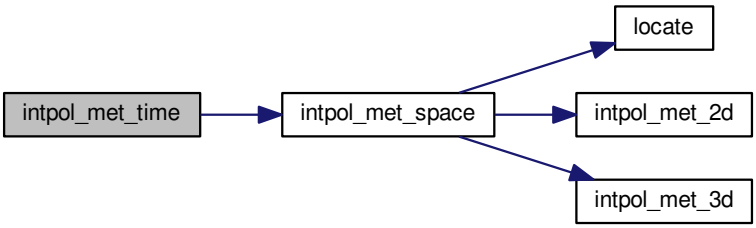
5.13.2.14 void intpol_met_time (met_t * *met0*, met_t * *met1*, double *ts*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

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

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 341 of file libtrac.c.

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

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

Find array index.

Definition at line 376 of file libtrac.c.

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

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

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

Read atmospheric data.

Definition at line 408 of file libtrac.c.

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

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

Read control parameters.

Definition at line 458 of file libtrac.c.

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

```
00549     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00550       ctl->qnt_tsts = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00553       ctl->qnt_tnat = iq;
00554       sprintf(ctl->qnt_unit[iq], "K");
00555     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00556       ctl->qnt_gw_var = iq;
00557       sprintf(ctl->qnt_unit[iq], "K^2");
00558     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00559       ctl->qnt_stat = iq;
00560       sprintf(ctl->qnt_unit[iq], "-");
00561     } else
00562       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00563   }
00564
00565   /* Time steps of simulation... */
00566   ctl->direction =
00567     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00568   if (ctl->direction != -1 && ctl->direction != 1)
00569     ERRMSG("Set DIRECTION to -1 or 1!");
00570   ctl->t_start =
00571     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00572   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00573   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00574
00575   /* Meteorological data... */
00576   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00577   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00578   if (ctl->met_np > EP)
00579     ERRMSG("Too many levels!");
00580   for (ip = 0; ip < ctl->met_np; ip++)
00581     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00582
00583   /* Isosurface parameters... */
00584   ctl->isosurf
00585     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00586   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00587
00588   /* Diffusion parameters... */
00589   ctl->turb_dx_trop
00590     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00591   ctl->turb_dx_strat
00592     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00593   ctl->turb_dz_trop
00594     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00595   ctl->turb_dz_strat
00596     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00597   ctl->turb_meso =
00598     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00599
00600   /* Life time of particles... */
00601   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00602   ctl->tdec_strat =
00603     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00604
00605   /* PSC analysis... */
00606   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00607   ctl->psc_hno3 =
00608     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00609
00610   /* Gravity wave analysis... */
00611   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
00612
00613   /* Output of atmospheric data... */
00614   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00615   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00616   ctl->atm_dt_out =
00617     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00618   ctl->atm_filter =
00619     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00620
00621   /* Output of CSI data... */
00622   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00623   ctl->csi_dt_out =
00624     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00625   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00626            ctl->csi_obsfile);
00627   ctl->csi_obsmin =
00628     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00629   ctl->csi_modmin =
00630     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00631   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00632   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
```

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

Here is the call graph for this function:

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

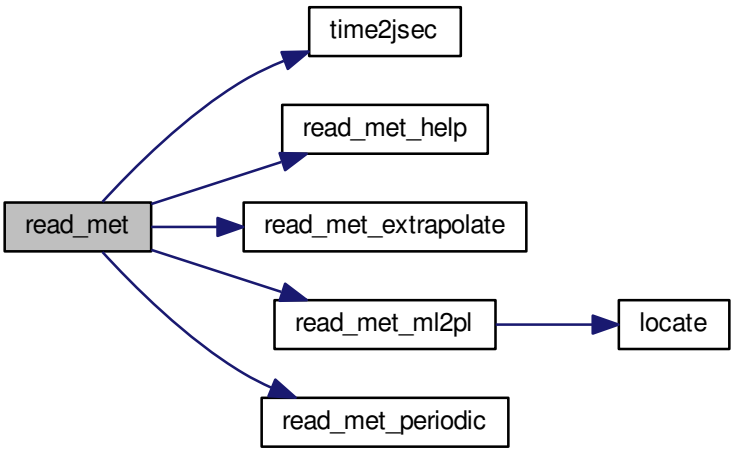
Read meteorological data file.

Definition at line 703 of file libtrac.c.

```
00706                {
00707
00708   char tstr[10];
00709
00710   static float help[EX * EY];
00711
00712   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00713
00714   size_t np, nx, ny;
00715
00716   /* Write info... */
00717   printf("Read meteorological data: %s\n", filename);
00718
00719   /* Get time from filename... */
00720   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00721   year = atoi(tstr);
00722   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00723   mon = atoi(tstr);
00724   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00725   day = atoi(tstr);
00726   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00727   hour = atoi(tstr);
00728   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00729
00730   /* Open netCDF file... */
00731   NC(nc_open(filename, NC_NOWRITE, &ncid));
00732
00733   /* Get dimensions... */
00734   NC(nc_inq_dimid(ncid, "lon", &dimid));
00735   NC(nc_inq_dimlen(ncid, dimid, &nx));
00736   if (nx > EX)
00737     ERRMSG("Too many longitudes!");
00738
00739   NC(nc_inq_dimid(ncid, "lat", &dimid));
00740   NC(nc_inq_dimlen(ncid, dimid, &ny));
00741   if (ny > EY)
00742     ERRMSG("Too many latitudes!");
00743
00744   NC(nc_inq_dimid(ncid, "lev", &dimid));
00745   NC(nc_inq_dimlen(ncid, dimid, &np));
00746   if (np > EP)
00747     ERRMSG("Too many levels!");
00748
00749   /* Store dimensions... */
00750   met->np = (int) np;
00751   met->nx = (int) nx;
00752   met->ny = (int) ny;
00753
00754   /* Get horizontal grid... */
00755   NC(nc_inq_varid(ncid, "lon", &varid));
00756   NC(nc_get_var_double(ncid, varid, met->lon));
00757   NC(nc_inq_varid(ncid, "lat", &varid));
00758   NC(nc_get_var_double(ncid, varid, met->lat));
00759
00760   /* Read meteorological data... */
00761   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00762   read_met_help(ncid, "u", "U", met, met->u, 1.0);
```

```
00763   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00764   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00765   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00766   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00767
00768   /* Meteo data on pressure levels... */
00769   if (ctl->met_np <= 0) {
00770
00771     /* Read pressure levels from file... */
00772     NC(nc_inq_varid(ncid, "lev", &varid));
00773     NC(nc_get_var_double(ncid, varid, met->p));
00774     for (ip = 0; ip < met->np; ip++)
00775       met->p[ip] /= 100.;
00776
00777     /* Extrapolate data for lower boundary... */
00778     read_met_extrapolate(met);
00779   }
00780
00781   /* Meteo data on model levels... */
00782   else {
00783
00784     /* Read pressure data from file... */
00785     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00786
00787     /* Interpolate from model levels to pressure levels... */
00788     read_met_ml2pl(ctl, met, met->t);
00789     read_met_ml2pl(ctl, met, met->u);
00790     read_met_ml2pl(ctl, met, met->v);
00791     read_met_ml2pl(ctl, met, met->w);
00792     read_met_ml2pl(ctl, met, met->h2o);
00793     read_met_ml2pl(ctl, met, met->o3);
00794
00795     /* Set pressure levels... */
00796     met->np = ctl->met_np;
00797     for (ip = 0; ip < met->np; ip++)
00798       met->p[ip] = ctl->met_p[ip];
00799   }
00800
00801   /* Check ordering of pressure levels... */
00802   for (ip = 1; ip < met->np; ip++)
00803     if (met->p[ip - 1] < met->p[ip])
00804       ERRMSG("Pressure levels must be descending!");
00805
00806   /* Read surface pressure... */
00807   if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00808     NC(nc_get_var_float(ncid, varid, help));
00809     for (iy = 0; iy < met->ny; iy++)
00810       for (ix = 0; ix < met->nx; ix++)
00811         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00812   } else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00813     NC(nc_get_var_float(ncid, varid, help));
00814     for (iy = 0; iy < met->ny; iy++)
00815       for (ix = 0; ix < met->nx; ix++)
00816         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00817   } else
00818     for (ix = 0; ix < met->nx; ix++)
00819       for (iy = 0; iy < met->ny; iy++)
00820         met->ps[ix][iy] = met->p[0];
00821
00822   /* Create periodic boundary conditions... */
00823   read_met_periodic(met);
00824
00825   /* Close file... */
00826   NC(nc_close(ncid));
00827 }
```

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 831 of file libtrac.c.

```
00832                {
00833
00834   int ip, ip0, ix, iy;
00835
00836   /* Loop over columns... */
00837   for (ix = 0; ix < met->nx; ix++)
00838     for (iy = 0; iy < met->ny; iy++) {
00839
00840       /* Find lowest valid data point... */
00841       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00842         if (!gsl_finite(met->t[ix][iy][ip0])
00843             || !gsl_finite(met->u[ix][iy][ip0])
00844             || !gsl_finite(met->v[ix][iy][ip0])
00845             || !gsl_finite(met->w[ix][iy][ip0]))
00846           break;
00847
00848       /* Extrapolate... */
00849       for (ip = ip0; ip >= 0; ip--) {
00850         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00851         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00852         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00853         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00854         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00855         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00856       }
00857     }
00858 }
```

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 862 of file libtrac.c.

```
00868             {
00869
00870   static float help[EX * EY * EP];
00871
00872   int ip, ix, iy, n = 0, varid;
00873
00874   /* Check if variable exists... */
00875   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00876     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00877       return;
00878
00879   /* Read data... */
00880   NC(nc_get_var_float(ncid, varid, help));
00881
00882   /* Copy and check data... */
00883   for (ip = 0; ip < met->np; ip++)
00884     for (iy = 0; iy < met->ny; iy++)
00885       for (ix = 0; ix < met->nx; ix++) {
00886         dest[ix][iy][ip] = scl * help[n++];
00887         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00888           dest[ix][iy][ip] = GSL_NAN;
00889       }
00890 }
```

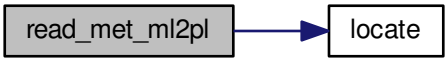
5.13.2.22 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 894 of file libtrac.c.

```
00897                         {
00898
00899   double aux[EP], p[EP], pt;
00900
00901   int ip, ip2, ix, iy;
00902
00903   /* Loop over columns... */
00904   for (ix = 0; ix < met->nx; ix++)
00905     for (iy = 0; iy < met->ny; iy++) {
00906
00907       /* Copy pressure profile... */
00908       for (ip = 0; ip < met->np; ip++)
00909         p[ip] = met->pl[ix][iy][ip];
00910
00911       /* Interpolate... */
00912       for (ip = 0; ip < ctl->met_np; ip++) {
00913         pt = ctl->met_p[ip];
00914         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00915           pt = p[0];
00916         else if ((pt > p[met->np - 1] && p[1] > p[0])
00917                  || (pt < p[met->np - 1] && p[1] < p[0]))
00918           pt = p[met->np - 1];
00919         ip2 = locate(p, met->np, pt);
00920         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00921                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00922       }
00923
00924       /* Copy data... */
00925       for (ip = 0; ip < ctl->met_np; ip++)
00926         var[ix][iy][ip] = (float) aux[ip];
00927     }
00928 }
```

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 932 of file libtrac.c.

```
00933               {
00934
00935   int ip, iy;
00936
00937   /* Check longitudes... */
00938   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00939              + met->lon[1] - met->lon[0] - 360) < 0.01))
00940     return;
00941
00942   /* Increase longitude counter... */
00943   if ((++met->nx) > EX)
00944     ERRMSG("Cannot create periodic boundary conditions!");
00945
00946   /* Set longitude... */
00947   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00948
00949   /* Loop over latitudes and pressure levels... */
00950   for (iy = 0; iy < met->ny; iy++)
00951     for (ip = 0; ip < met->np; ip++) {
00952       met->ps[met->nx - 1][iy] = met->ps[0][iy];
00953       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
00954       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00955       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00956       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00957       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00958       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00959     }
00960 }
```

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 964 of file libtrac.c.

```
00971               {
00972
00973   FILE *in = NULL;
00974
00975   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00976     msg[LEN], rvarname[LEN], rval[LEN];
00977
00978   int contain = 0, i;
00979
00980   /* Open file... */
00981   if (filename[strlen(filename) - 1] != '-')
00982     if (!(in = fopen(filename, "r")))
00983       ERRMSG("Cannot open file!");
00984
00985   /* Set full variable name... */
00986   if (arridx >= 0) {
00987     sprintf(fullname1, "%s[%d]", varname, arridx);
00988     sprintf(fullname2, "%s[*]", varname);
00989   } else {
00990     sprintf(fullname1, "%s", varname);
00991     sprintf(fullname2, "%s", varname);
00992   }
00993
00994   /* Read data... */
00995   if (in != NULL)
00996     while (fgets(line, LEN, in))
00997       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00998         if (strcasecmp(rvarname, fullname1) == 0 ||
00999             strcasecmp(rvarname, fullname2) == 0) {
01000           contain = 1;
01001           break;
01002         }
01003   for (i = 1; i < argc - 1; i++)
```

```
01004     if (strcasecmp(argv[i], fullname1) == 0 ||
01005         strcasecmp(argv[i], fullname2) == 0) {
01006       sprintf(rval, "%s", argv[i + 1]);
01007       contain = 1;
01008       break;
01009     }
01010
01011   /* Close file... */
01012   if (in != NULL)
01013     fclose(in);
01014
01015   /* Check for missing variables... */
01016   if (!contain) {
01017     if (strlen(defvalue) > 0)
01018       sprintf(rval, "%s", defvalue);
01019     else {
01020       sprintf(msg, "Missing variable %s!\n", fullname1);
01021       ERRMSG(msg);
01022     }
01023   }
01024
01025   /* Write info... */
01026   printf("%s = %s\n", fullname1, rval);
01027
01028   /* Return values... */
01029   if (value != NULL)
01030     sprintf(value, "%s", rval);
01031   return atof(rval);
01032 }
```

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 1036 of file libtrac.c.

```
01044                 {
01045
01046   struct tm t0, t1;
01047
01048   t0.tm_year = 100;
01049   t0.tm_mon = 0;
01050   t0.tm_mday = 1;
01051   t0.tm_hour = 0;
01052   t0.tm_min = 0;
01053   t0.tm_sec = 0;
01054
01055   t1.tm_year = year - 1900;
01056   t1.tm_mon = mon - 1;
01057   t1.tm_mday = day;
01058   t1.tm_hour = hour;
01059   t1.tm_min = min;
01060   t1.tm_sec = sec;
01061
01062   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01063 }
```

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

Measure wall-clock time.

Definition at line 1067 of file libtrac.c.

```
01070             {
01071
01072   static double starttime[NTIMER], runtime[NTIMER];
01073
01074   /* Check id... */
01075   if (id < 0 || id >= NTIMER)
01076     ERRMSG("Too many timers!");
01077
01078   /* Start timer... */
01079   if (mode == 1) {
01080     if (starttime[id] <= 0)
```

```
01081       starttime[id] = omp_get_wtime();
01082     else
01083       ERRMSG("Timer already started!");
01084   }
01085
01086   /* Stop timer... */
01087   else if (mode == 2) {
01088     if (starttime[id] > 0) {
01089       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01090       starttime[id] = -1;
01091     } else
01092       ERRMSG("Timer not started!");
01093   }
01094
01095   /* Print timer... */
01096   else if (mode == 3)
01097     printf("%s = %g s\n", name, runtime[id]);
01098 }
```

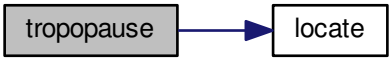
5.13.2.27 double tropopause (double *t*, double *lat*)

Definition at line 1102 of file libtrac.c.

```
01104               {
01105
01106   static double doys[12]
01107   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01108
01109   static double lats[73]
01110     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01111     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01112     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01113     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01114     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01115     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01116     75, 77.5, 80, 82.5, 85, 87.5, 90
01117   };
01118
01119   static double tps[12][73]
01120     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01121          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01122          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01123          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01124          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01125          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01126          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01127          275.3, 275.6, 275.4, 274.1, 273.5},
01128   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01129    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01130    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01131    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01132    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01133    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01134    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01135    287.5, 286.2, 285.8},
01136   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01137    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01138    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01139    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01140    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01141    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01142    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01143    304.3, 304.9, 306, 306.6, 306.2, 306},
01144   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01145    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01146    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01147    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01148    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01149    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01150    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01151    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01152   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01153    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01154    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01155    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01156    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01157    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01158    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01159    325.3, 325.8, 325.8},
01160   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01161    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
```

```
01162     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01163     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01164     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01165     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01166     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01167     308.5, 312.2, 313.1, 313.3},
01168    {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01169     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01170     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01171     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01172     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01173     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01174     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01175     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01176    {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01177     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01178     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01179     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01180     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01181     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01182     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01183     278.2, 282.6, 287.4, 290.9, 292.5, 293},
01184    {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01185     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01186     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01187     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01188     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01189     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01190     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01191     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01192    {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01193     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01194     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01195     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01196     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01197     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01198     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01199     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01200     305.1},
01201    {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01202     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01203     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01204     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01205     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01206     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01207     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01208     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01209    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01210     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01211     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01212     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01213     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01214     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01215     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01216     281.7, 281.1, 281.2}
01217    };
01218
01219    double doy, p0, p1, pt;
01220
01221    int imon, ilat;
01222
01223    /* Get day of year... */
01224    doy = fmod(t / 86400., 365.25);
01225    while (doy < 0)
01226      doy += 365.25;
01227
01228    /* Get indices... */
01229    imon = locate(doys, 12, doy);
01230    ilat = locate(lats, 73, lat);
01231
01232    /* Get tropopause pressure... */
01233    p0 = LIN(lats[ilat], tps[imon][ilat],
01234             lats[ilat + 1], tps[imon][ilat + 1], lat);
01235    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01236             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01237    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01238
01239    /* Return tropopause pressure... */
01240    return pt;
01241  }
```

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 1245 of file libtrac.c.

```
01249            {
01250
01251   FILE *in, *out;
01252
01253   char line[LEN];
01254
01255   double r, t0, t1;
01256
01257   int ip, iq, year, mon, day, hour, min, sec;
01258
01259   /* Set time interval for output... */
01260   t0 = t - 0.5 * ctl->dt_mod;
01261   t1 = t + 0.5 * ctl->dt_mod;
01262
01263   /* Check if gnuplot output is requested... */
01264   if (ctl->atm_gpfile[0] != '-') {
01265
01266     /* Write info... */
01267     printf("Plot atmospheric data: %s.png\n", filename);
01268
01269     /* Create gnuplot pipe... */
01270     if (!(out = popen("gnuplot", "w")))
01271       ERRMSG("Cannot create pipe to gnuplot!");
01272
01273     /* Set plot filename... */
01274     fprintf(out, "set out \"%s.png\"\n", filename);
01275
01276     /* Set time string... */
01277     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01278     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01279             year, mon, day, hour, min);
01280
01281     /* Dump gnuplot file to pipe... */
01282     if (!(in = fopen(ctl->atm_gpfile, "r")))
01283       ERRMSG("Cannot open file!");
01284     while (fgets(line, LEN, in))
01285       fprintf(out, "%s", line);
01286     fclose(in);
01287   }
01288
01289   else {
01290
01291     /* Write info... */
01292     printf("Write atmospheric data: %s\n", filename);
01293
01294     /* Create file... */
01295     if (!(out = fopen(filename, "w")))
01296       ERRMSG("Cannot create file!");
01297   }
01298
01299   /* Write header... */
01300   fprintf(out,
01301           "# $1 = time [s]\n"
01302           "# $2 = altitude [km]\n"
01303           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01304   for (iq = 0; iq < ctl->nq; iq++)
01305     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
```

```
01306             ctl->qnt_unit[iq]);
01307   fprintf(out, "\n");
01308
01309   /* Write data... */
01310   for (ip = 0; ip < atm->np; ip++) {
01311
01312     /* Check time... */
01313     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01314       continue;
01315
01316     /* Write output... */
01317     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01318             atm->lon[ip], atm->lat[ip]);
01319     for (iq = 0; iq < ctl->nq; iq++) {
01320       fprintf(out, " ");
01321       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01322     }
01323     fprintf(out, "\n");
01324   }
01325
01326   /* Close file... */
01327   fclose(out);
01328 }
```

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 1332 of file libtrac.c.

```
01336             {
01337
01338   static FILE *in, *out;
01339
01340   static char line[LEN];
01341
01342   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01343     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01344
01345   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01346
01347   /* Init... */
01348   if (!init) {
01349     init = 1;
01350
01351     /* Check quantity index for mass... */
01352     if (ctl->qnt_m < 0)
01353       ERRMSG("Need quantity mass to analyze CSI!");
01354
01355     /* Open observation data file... */
01356     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01357     if (!(in = fopen(ctl->csi_obsfile, "r")))
01358       ERRMSG("Cannot open file!");
01359
01360     /* Create new file... */
01361     printf("Write CSI data: %s\n", filename);
01362     if (!(out = fopen(filename, "w")))
01363       ERRMSG("Cannot create file!");
01364
01365     /* Write header... */
```

```
01366     fprintf(out,
01367             "# $1 = time [s]\n"
01368             "# $2 = number of hits (cx)\n"
01369             "# $3 = number of misses (cy)\n"
01370             "# $4 = number of false alarms (cz)\n"
01371             "# $5 = number of observations (cx + cy)\n"
01372             "# $6 = number of forecasts (cx + cz)\n"
01373             "# $7 = bias (forecasts/observations) [%%]\n"
01374             "# $8 = probability of detection (POD) [%%]\n"
01375             "# $9 = false alarm rate (FAR) [%%]\n"
01376             "# $10 = critical success index (CSI) [%%]\n\n");
01377   }
01378
01379   /* Set time interval... */
01380   t0 = t - 0.5 * ctl->dt_mod;
01381   t1 = t + 0.5 * ctl->dt_mod;
01382
01383   /* Initialize grid cells... */
01384   for (ix = 0; ix < ctl->csi_nx; ix++)
01385     for (iy = 0; iy < ctl->csi_ny; iy++)
01386       for (iz = 0; iz < ctl->csi_nz; iz++)
01387         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01388
01389   /* Read data... */
01390   while (fgets(line, LEN, in)) {
01391
01392     /* Read data... */
01393     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01394         5)
01395       continue;
01396
01397     /* Check time... */
01398     if (rt < t0)
01399       continue;
01400     if (rt > t1)
01401       break;
01402
01403     /* Calculate indices... */
01404     ix = (int) ((rlon - ctl->csi_lon0)
01405                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01406     iy = (int) ((rlat - ctl->csi_lat0)
01407                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01408     iz = (int) ((rz - ctl->csi_z0)
01409                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01410
01411     /* Check indices... */
01412     if (ix < 0 || ix >= ctl->csi_nx ||
01413         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01414       continue;
01415
01416     /* Get mean observation index... */
01417     obsmean[ix][iy][iz] += robs;
01418     obscount[ix][iy][iz]++;
01419   }
01420
01421   /* Analyze model data... */
01422   for (ip = 0; ip < atm->np; ip++) {
01423
01424     /* Check time... */
01425     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01426       continue;
01427
01428     /* Get indices... */
01429     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01430                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01431     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01432                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01433     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01434                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01435
01436     /* Check indices... */
01437     if (ix < 0 || ix >= ctl->csi_nx ||
01438         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01439       continue;
01440
01441     /* Get total mass in grid cell... */
01442     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01443   }
01444
01445   /* Analyze all grid cells... */
01446   for (ix = 0; ix < ctl->csi_nx; ix++)
01447     for (iy = 0; iy < ctl->csi_ny; iy++)
01448       for (iz = 0; iz < ctl->csi_nz; iz++) {
01449
01450         /* Calculate mean observation index... */
01451         if (obscount[ix][iy][iz] > 0)
01452           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
```

```
01453
01454         /* Calculate column density... */
01455         if (modmean[ix][iy][iz] > 0) {
01456           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01457           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01458           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01459           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01460             * cos(lat * M_PI / 180.);
01461           modmean[ix][iy][iz] /= (1e6 * area);
01462         }
01463
01464         /* Calculate CSI... */
01465         if (obscount[ix][iy][iz] > 0) {
01466           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01467               modmean[ix][iy][iz] >= ctl->csi_modmin)
01468             cx++;
01469           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01470                    modmean[ix][iy][iz] < ctl->csi_modmin)
01471             cy++;
01472           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01473                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01474             cz++;
01475         }
01476       }
01477
01478   /* Write output... */
01479   if (fmod(t, ctl->csi_dt_out) == 0) {
01480
01481     /* Write... */
01482     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01483             t, cx, cy, cz, cx + cy, cx + cz,
01484             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01485             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01486             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01487             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01488
01489     /* Set counters to zero... */
01490     cx = cy = cz = 0;
01491   }
01492
01493   /* Close file... */
01494   if (t == ctl->t_stop)
01495     fclose(out);
01496 }
```

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

Write ensemble data.

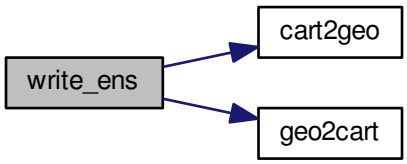
Definition at line 1500 of file libtrac.c.

```
01504             {
01505
01506   static FILE *out;
01507
01508   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01509     t0, t1, x[NENS][3], xm[3];
01510
01511   static int init, ip, iq;
01512
01513   static size_t i, n;
01514
01515   /* Init... */
01516   if (!init) {
01517     init = 1;
01518
01519     /* Check quantities... */
01520     if (ctl->qnt_ens < 0)
01521       ERRMSG("Missing ensemble IDs!");
01522
01523     /* Create new file... */
01524     printf("Write ensemble data: %s\n", filename);
01525     if (!(out = fopen(filename, "w")))
01526       ERRMSG("Cannot create file!");
01527
01528     /* Write header... */
01529     fprintf(out,
01530             "# $1 = time [s]\n"
01531             "# $2 = altitude [km]\n"
01532             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
```

```
01533     for (iq = 0; iq < ctl->nq; iq++)
01534       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01535               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01536     for (iq = 0; iq < ctl->nq; iq++)
01537       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01538               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01539     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01540   }
01541
01542   /* Set time interval... */
01543   t0 = t - 0.5 * ctl->dt_mod;
01544   t1 = t + 0.5 * ctl->dt_mod;
01545
01546   /* Init... */
01547   ens = GSL_NAN;
01548   n = 0;
01549
01550   /* Loop over air parcels... */
01551   for (ip = 0; ip < atm->np; ip++) {
01552
01553     /* Check time... */
01554     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01555       continue;
01556
01557     /* Check ensemble id... */
01558     if (atm->q[ctl->qnt_ens][ip] != ens) {
01559
01560       /* Write results... */
01561       if (n > 0) {
01562
01563         /* Get mean position... */
01564         xm[0] = xm[1] = xm[2] = 0;
01565         for (i = 0; i < n; i++) {
01566           xm[0] += x[i][0] / (double) n;
01567           xm[1] += x[i][1] / (double) n;
01568           xm[2] += x[i][2] / (double) n;
01569         }
01570         cart2geo(xm, &dummy, &lon, &lat);
01571         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01572                 lat);
01573
01574         /* Get quantity statistics... */
01575         for (iq = 0; iq < ctl->nq; iq++) {
01576           fprintf(out, " ");
01577           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01578         }
01579         for (iq = 0; iq < ctl->nq; iq++) {
01580           fprintf(out, " ");
01581           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01582         }
01583         fprintf(out, " %lu\n", n);
01584       }
01585
01586       /* Init new ensemble... */
01587       ens = atm->q[ctl->qnt_ens][ip];
01588       n = 0;
01589     }
01590
01591     /* Save data... */
01592     p[n] = atm->p[ip];
01593     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01594     for (iq = 0; iq < ctl->nq; iq++)
01595       q[iq][n] = atm->q[iq][ip];
01596     if ((++n) >= NENS)
01597       ERRMSG("Too many data points!");
01598   }
01599
01600   /* Write results... */
01601   if (n > 0) {
01602
01603     /* Get mean position... */
01604     xm[0] = xm[1] = xm[2] = 0;
01605     for (i = 0; i < n; i++) {
01606       xm[0] += x[i][0] / (double) n;
01607       xm[1] += x[i][1] / (double) n;
01608       xm[2] += x[i][2] / (double) n;
01609     }
01610     cart2geo(xm, &dummy, &lon, &lat);
01611     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01612
01613     /* Get quantity statistics... */
01614     for (iq = 0; iq < ctl->nq; iq++) {
01615       fprintf(out, " ");
01616       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01617     }
01618     for (iq = 0; iq < ctl->nq; iq++) {
01619       fprintf(out, " ");
```

```
01620       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01621     }
01622     fprintf(out, " %lu\n", n);
01623   }
01624
01625   /* Close file... */
01626   if (t == ctl->t_stop)
01627     fclose(out);
01628 }
```

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.

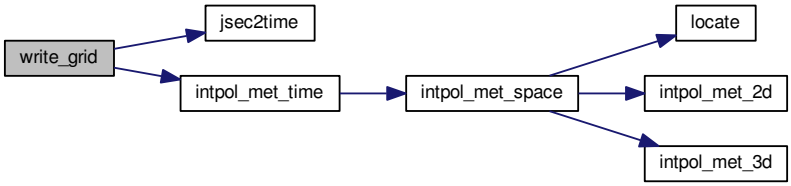
Definition at line 1632 of file libtrac.c.

```
01638             {
01639
01640   FILE *in, *out;
01641
01642   char line[LEN];
01643
01644   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01645     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01646
01647   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01648
01649   /* Check dimensions... */
01650   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01651     ERRMSG("Grid dimensions too large!");
01652
01653   /* Check quantity index for mass... */
01654   if (ctl->qnt_m < 0)
01655     ERRMSG("Need quantity mass to write grid data!");
01656
01657   /* Set time interval for output... */
01658   t0 = t - 0.5 * ctl->dt_mod;
01659   t1 = t + 0.5 * ctl->dt_mod;
01660
01661   /* Set grid box size... */
01662   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01663   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01664   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01665
01666   /* Initialize grid... */
01667   for (ix = 0; ix < ctl->grid_nx; ix++)
01668     for (iy = 0; iy < ctl->grid_ny; iy++)
01669       for (iz = 0; iz < ctl->grid_nz; iz++)
01670         grid_m[ix][iy][iz] = 0;
01671
01672   /* Average data... */
01673   for (ip = 0; ip < atm->np; ip++)
01674     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01675
01676       /* Get index... */
01677       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
```

```
01678       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01679       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01680
01681       /* Check indices... */
01682       if (ix < 0 || ix >= ctl->grid_nx ||
01683           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01684         continue;
01685
01686       /* Add mass... */
01687       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01688     }
01689
01690   /* Check if gnuplot output is requested... */
01691   if (ctl->grid_gpfile[0] != '-') {
01692
01693     /* Write info... */
01694     printf("Plot grid data: %s.png\n", filename);
01695
01696     /* Create gnuplot pipe... */
01697     if (!(out = popen("gnuplot", "w")))
01698       ERRMSG("Cannot create pipe to gnuplot!");
01699
01700     /* Set plot filename... */
01701     fprintf(out, "set out \"%s.png\"\n", filename);
01702
01703     /* Set time string... */
01704     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01705     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01706             year, mon, day, hour, min);
01707
01708     /* Dump gnuplot file to pipe... */
01709     if (!(in = fopen(ctl->grid_gpfile, "r")))
01710       ERRMSG("Cannot open file!");
01711     while (fgets(line, LEN, in))
01712       fprintf(out, "%s", line);
01713     fclose(in);
01714   }
01715
01716   else {
01717
01718     /* Write info... */
01719     printf("Write grid data: %s\n", filename);
01720
01721     /* Create file... */
01722     if (!(out = fopen(filename, "w")))
01723       ERRMSG("Cannot create file!");
01724   }
01725
01726   /* Write header... */
01727   fprintf(out,
01728           "# $1 = time [s]\n"
01729           "# $2 = altitude [km]\n"
01730           "# $3 = longitude [deg]\n"
01731           "# $4 = latitude [deg]\n"
01732           "# $5 = surface area [km^2]\n"
01733           "# $6 = layer width [km]\n"
01734           "# $7 = temperature [K]\n"
01735           "# $8 = column density [kg/m^2]\n"
01736           "# $9 = mass mixing ratio [1]\n\n");
01737
01738   /* Write data... */
01739   for (ix = 0; ix < ctl->grid_nx; ix++) {
01740     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01741       fprintf(out, "\n");
01742     for (iy = 0; iy < ctl->grid_ny; iy++) {
01743       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01744         fprintf(out, "\n");
01745       for (iz = 0; iz < ctl->grid_nz; iz++)
01746         if (!ctl->grid_sparse
01747             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01748
01749           /* Set coordinates... */
01750           z = ctl->grid_z0 + dz * (iz + 0.5);
01751           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01752           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01753
01754           /* Get pressure and temperature... */
01755           press = P(z);
01756           intpol_met_time(met0, met1, t, press, lon, lat,
01757                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01758
01759           /* Calculate surface area... */
01760           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01761             * cos(lat * M_PI / 180.);
01762
01763           /* Calculate column density... */
01764           cd = grid_m[ix][iy][iz] / (1e6 * area);
```

```
01765
01766           /* Calculate mass mixing ratio... */
01767           rho_air = 100. * press / (287.058 * temp);
01768           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01769
01770           /* Write output... */
01771           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01772                   t, z, lon, lat, area, dz, temp, cd, mmr);
01773         }
01774     }
01775   }
01776
01777   /* Close file... */
01778   fclose(out);
01779 }
```

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.

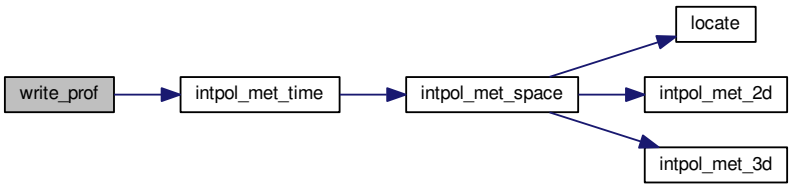
Definition at line 1783 of file libtrac.c.

```
01789           {
01790
01791   static FILE *in, *out;
01792
01793   static char line[LEN];
01794
01795   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01796     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01797     press, temp, rho_air, mmr, h2o, o3;
01798
01799   static int init, obscount[GX][GY], ip, ix, iy, iz;
01800
01801   /* Init... */
01802   if (!init) {
01803     init = 1;
01804
01805     /* Check quantity index for mass... */
01806     if (ctl->qnt_m < 0)
01807       ERRMSG("Need quantity mass!");
01808
01809     /* Check dimensions... */
01810     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01811       ERRMSG("Grid dimensions too large!");
01812
01813     /* Open observation data file... */
01814     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01815     if (!(in = fopen(ctl->prof_obsfile, "r")))
01816       ERRMSG("Cannot open file!");
01817
01818     /* Create new file... */
01819     printf("Write profile data: %s\n", filename);
01820     if (!(out = fopen(filename, "w")))
01821       ERRMSG("Cannot create file!");
01822
01823     /* Write header... */
01824     fprintf(out,
```

```
01825             "# $1  = time [s]\n"
01826             "# $2  = altitude [km]\n"
01827             "# $3  = longitude [deg]\n"
01828             "# $4  = latitude [deg]\n"
01829             "# $5  = pressure [hPa]\n"
01830             "# $6  = temperature [K]\n"
01831             "# $7  = mass mixing ratio [1]\n"
01832             "# $8  = H2O volume mixing ratio [1]\n"
01833             "# $9  = O3 volume mixing ratio [1]\n"
01834             "# $10 = mean BT index [K]\n");
01835
01836     /* Set grid box size... */
01837     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01838     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01839     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01840   }
01841
01842   /* Set time interval... */
01843   t0 = t - 0.5 * ctl->dt_mod;
01844   t1 = t + 0.5 * ctl->dt_mod;
01845
01846   /* Initialize... */
01847   for (ix = 0; ix < ctl->prof_nx; ix++)
01848     for (iy = 0; iy < ctl->prof_ny; iy++) {
01849       obsmean[ix][iy] = 0;
01850       obscount[ix][iy] = 0;
01851       tmean[ix][iy] = 0;
01852       for (iz = 0; iz < ctl->prof_nz; iz++)
01853         mass[ix][iy][iz] = 0;
01854     }
01855
01856   /* Read data... */
01857   while (fgets(line, LEN, in)) {
01858
01859     /* Read data... */
01860     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01861       continue;
01862
01863     /* Check time... */
01864     if (rt < t0)
01865       continue;
01866     if (rt > t1)
01867       break;
01868
01869     /* Calculate indices... */
01870     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01871     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01872
01873     /* Check indices... */
01874     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01875       continue;
01876
01877     /* Get mean observation index... */
01878     obsmean[ix][iy] += robs;
01879     tmean[ix][iy] += rt;
01880     obscount[ix][iy]++;
01881   }
01882
01883   /* Analyze model data... */
01884   for (ip = 0; ip < atm->np; ip++) {
01885
01886     /* Check time... */
01887     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01888       continue;
01889
01890     /* Get indices... */
01891     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01892     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01893     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01894
01895     /* Check indices... */
01896     if (ix < 0 || ix >= ctl->prof_nx ||
01897         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01898       continue;
01899
01900     /* Get total mass in grid cell... */
01901     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01902   }
01903
01904   /* Extract profiles... */
01905   for (ix = 0; ix < ctl->prof_nx; ix++)
01906     for (iy = 0; iy < ctl->prof_ny; iy++)
01907       if (obscount[ix][iy] > 0) {
01908
01909         /* Write output... */
01910         fprintf(out, "\n");
01911
```

```
01912         /* Loop over altitudes... */
01913         for (iz = 0; iz < ctl->prof_nz; iz++) {
01914
01915           /* Set coordinates... */
01916           z = ctl->prof_z0 + dz * (iz + 0.5);
01917           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01918           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01919
01920           /* Get meteorological data... */
01921           press = P(z);
01922           intpol_met_time(met0, met1, t, press, lon, lat,
01923                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01924
01925           /* Calculate mass mixing ratio... */
01926           rho_air = 100. * press / (287.058 * temp);
01927           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
01928             * cos(lat * M_PI / 180.);
01929           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01930
01931           /* Write output... */
01932           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01933                   tmean[ix][iy] / obscount[ix][iy],
01934                   z, lon, lat, press, temp, mmr, h2o, o3,
01935                   obsmean[ix][iy] / obscount[ix][iy]);
01936         }
01937       }
01938
01939   /* Close file... */
01940   if (t == ctl->t_stop)
01941     fclose(out);
01942 }
```

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*)

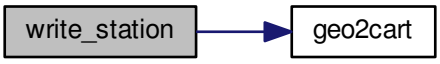
Write station data.

Definition at line 1946 of file libtrac.c.

```
01950             {
01951
01952   static FILE *out;
01953
01954   static double rmax2, t0, t1, x0[3], x1[3];
01955
01956   static int init, ip, iq;
01957
01958   /* Init... */
01959   if (!init) {
01960     init = 1;
01961
01962     /* Write info... */
01963     printf("Write station data: %s\n", filename);
01964
01965     /* Create new file... */
01966     if (!(out = fopen(filename, "w")))
01967       ERRMSG("Cannot create file!");
01968
01969     /* Write header... */
```

```
01970     fprintf(out,
01971             "# $1 = time [s]\n"
01972             "# $2 = altitude [km]\n"
01973             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01974     for (iq = 0; iq < ctl->nq; iq++)
01975       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01976               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01977     fprintf(out, "\n");
01978
01979     /* Set geolocation and search radius... */
01980     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01981     rmax2 = gsl_pow_2(ctl->stat_r);
01982   }
01983
01984   /* Set time interval for output... */
01985   t0 = t - 0.5 * ctl->dt_mod;
01986   t1 = t + 0.5 * ctl->dt_mod;
01987
01988   /* Loop over air parcels... */
01989   for (ip = 0; ip < atm->np; ip++) {
01990
01991     /* Check time... */
01992     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01993       continue;
01994
01995     /* Check station flag... */
01996     if (ctl->qnt_stat >= 0)
01997       if (atm->q[ctl->qnt_stat][ip])
01998         continue;
01999
02000     /* Get Cartesian coordinates... */
02001     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02002
02003     /* Check horizontal distance... */
02004     if (DIST2(x0, x1) > rmax2)
02005       continue;
02006
02007     /* Set station flag... */
02008     if (ctl->qnt_stat >= 0)
02009       atm->q[ctl->qnt_stat][ip] = 1;
02010
02011     /* Write data... */
02012     fprintf(out, "%.2f %g %g %g",
02013             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02014     for (iq = 0; iq < ctl->nq; iq++) {
02015       fprintf(out, " ");
02016       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02017     }
02018     fprintf(out, "\n");
02019   }
02020
02021   /* Close file... */
02022   if (t == ctl->t_stop)
02023     fclose(out);
02024 }
```

Here is the call graph for this function:



5.14 libtrac.h

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

```
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00034 #include <ctype.h>
00035 #include <gsl/gsl_const_mksa.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_statistics.h>
00041 #include <math.h>
00042 #include <netcdf.h>
00043 #include <omp.h>
00044 #include <stdio.h>
00045 #include <stdlib.h>
00046 #include <string.h>
00047 #include <time.h>
00048 #include <sys/time.h>
00049
00050 /* ------------------------------------------------------------
00051    Macros...
00052    ------------------------------------------------------------ */
00053
00055 #define ALLOC(ptr, type, n)                             \
00056   if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)      \
00057     ERRMSG("Out of memory!");
00058
00060 #define DIST(a, b) sqrt(DIST2(a, b))
00061
00063 #define DIST2(a, b)                                                     \
00064   ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00065
00067 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00068
00070 #define ERRMSG(msg) {                                                   \
00071     printf("\nError (%s, %s, l%d): %s\n\n",                             \
00072            __FILE__, __func__, __LINE__, msg);                          \
00073     exit(EXIT_FAILURE);                                                 \
00074   }
00075
00077 #define LIN(x0, y0, x1, y1, x)                  \
00078   ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00079
00081 #define NC(cmd) {                                       \
00082     if((cmd)!=NC_NOERR)                                 \
00083       ERRMSG(nc_strerror(cmd));                         \
00084   }
00085
00087 #define NORM(a) sqrt(DOTP(a, a))
00088
00090 #define PRINT(format, var)                                              \
00091   printf("Print (%s, %s, l%d): %s= "format"\n",                         \
00092          __FILE__, __func__, __LINE__, #var, var);
00093
00095 #define P(z) (P0*exp(-(z)/H0))
00096
00098 #define TOK(line, tok, format, var) {                                   \
00099     if(((tok)=strtok((line), " \t"))) {                                 \
00100       if(sscanf(tok, format, &(var))!=1) continue;                      \
00101     } else ERRMSG("Error while reading!");                              \
00102   }
00103
00105 #define Z(p) (H0*log(P0/(p)))
00106
00108 #define START_TIMER(id) timer(#id, id, 1)
00109
00111 #define STOP_TIMER(id) timer(#id, id, 2)
00112
00114 #define PRINT_TIMER(id) timer(#id, id, 3)
00115
00116 /* ------------------------------------------------------------
00117    Constants...
00118    ------------------------------------------------------------ */
00119
00121 #define G0 9.80665
00122
```

```
00124 #define H0 7.0
00125
00127 #define P0 1013.25
00128
00130 #define RE 6367.421
00131
00132 /* ------------------------------------------------------------
00133    Dimensions...
00134    ------------------------------------------------------------ */
00135
00137 #define LEN 5000
00138
00140 #define NP 10000000
00141
00143 #define NQ 12
00144
00146 #define EP 137
00147
00149 #define EX 1201
00150
00152 #define EY 601
00153
00155 #define GX 720
00156
00158 #define GY 360
00159
00161 #define GZ 100
00162
00164 #define NENS 2000
00165
00167 #define NTHREADS 128
00168
00170 #define NTIMER 20
00171
00172 /* ------------------------------------------------------------
00173    Structs...
00174    ------------------------------------------------------------ */
00175
00177 typedef struct {
00178
00180   int nq;
00181
00183   char qnt_name[NQ][LEN];
00184
00186   char qnt_unit[NQ][LEN];
00187
00189   char qnt_format[NQ][LEN];
00190
00192   int qnt_ens;
00193
00195   int qnt_m;
00196
00198   int qnt_rho;
00199
00201   int qnt_r;
00202
00204   int qnt_ps;
00205
00207   int qnt_p;
00208
00210   int qnt_t;
00211
00213   int qnt_u;
00214
00216   int qnt_v;
00217
00219   int qnt_w;
00220
00222   int qnt_h2o;
00223
00225   int qnt_o3;
00226
00228   int qnt_theta;
00229
00231   int qnt_pv;
00232
00234   int qnt_tice;
00235
00237   int qnt_tsts;
00238
00240   int qnt_tnat;
00241
00243   int qnt_stat;
00244
00246   int qnt_gw_u750;
00247
00249   int qnt_gw_v750;
```

```
00250
00252   int qnt_gw_sso;
00253
00255   int qnt_gw_var;
00256
00258   int direction;
00259
00261   double t_start;
00262
00264   double t_stop;
00265
00267   double dt_mod;
00268
00270   double dt_met;
00271
00273   int met_np;
00274
00276   double met_p[EP];
00277
00280   int isosurf;
00281
00283   char balloon[LEN];
00284
00286   double turb_dx_trop;
00287
00289   double turb_dx_strat;
00290
00292   double turb_dz_trop;
00293
00295   double turb_dz_strat;
00296
00298   double turb_meso;
00299
00301   double tdec_trop;
00302
00304   double tdec_strat;
00305
00307   double psc_h2o;
00308
00310   double psc_hno3;
00311
00313   char gw_basename[LEN];
00314
00316   char atm_basename[LEN];
00317
00319   char atm_gpfile[LEN];
00320
00322   double atm_dt_out;
00323
00325   int atm_filter;
00326
00328   char csi_basename[LEN];
00329
00331   double csi_dt_out;
00332
00334   char csi_obsfile[LEN];
00335
00337   double csi_obsmin;
00338
00340   double csi_modmin;
00341
00343   int csi_nz;
00344
00346   double csi_z0;
00347
00349   double csi_z1;
00350
00352   int csi_nx;
00353
00355   double csi_lon0;
00356
00358   double csi_lon1;
00359
00361   int csi_ny;
00362
00364   double csi_lat0;
00365
00367   double csi_lat1;
00368
00370   char grid_basename[LEN];
00371
00373   char grid_gpfile[LEN];
00374
00376   double grid_dt_out;
00377
00379   int grid_sparse;
00380
```

```
00382   int grid_nz;
00383
00385   double grid_z0;
00386
00388   double grid_z1;
00389
00391   int grid_nx;
00392
00394   double grid_lon0;
00395
00397   double grid_lon1;
00398
00400   int grid_ny;
00401
00403   double grid_lat0;
00404
00406   double grid_lat1;
00407
00409   char prof_basename[LEN];
00410
00412   char prof_obsfile[LEN];
00413
00415   int prof_nz;
00416
00418   double prof_z0;
00419
00421   double prof_z1;
00422
00424   int prof_nx;
00425
00427   double prof_lon0;
00428
00430   double prof_lon1;
00431
00433   int prof_ny;
00434
00436   double prof_lat0;
00437
00439   double prof_lat1;
00440
00442   char ens_basename[LEN];
00443
00445   char stat_basename[LEN];
00446
00448   double stat_lon;
00449
00451   double stat_lat;
00452
00454   double stat_r;
00455
00456 } ctl_t;
00457
00459 typedef struct {
00460
00462   int np;
00463
00465   double time[NP];
00466
00468   double p[NP];
00469
00471   double lon[NP];
00472
00474   double lat[NP];
00475
00477   double q[NQ][NP];
00478
00480   double up[NP];
00481
00483   double vp[NP];
00484
00486   double wp[NP];
00487
00488 } atm_t;
00489
00491 typedef struct {
00492
00494   double time;
00495
00497   int nx;
00498
00500   int ny;
00501
00503   int np;
00504
00506   double lon[EX];
00507
00509   double lat[EY];
```

```
00510
00512   double p[EP];
00513
00515   double ps[EX][EY];
00516
00518   float pl[EX][EY][EP];
00519
00521   float t[EX][EY][EP];
00522
00524   float u[EX][EY][EP];
00525
00527   float v[EX][EY][EP];
00528
00530   float w[EX][EY][EP];
00531
00533   float h2o[EX][EY][EP];
00534
00536   float o3[EX][EY][EP];
00537
00538 } met_t;
00539
00540 /* ------------------------------------------------------------
00541    Functions...
00542    ------------------------------------------------------------ */
00543
00545 void cart2geo(
00546   double *x,
00547   double *z,
00548   double *lon,
00549   double *lat);
00550
00552 double deg2dx(
00553   double dlon,
00554   double lat);
00555
00557 double deg2dy(
00558   double dlat);
00559
00561 double dp2dz(
00562   double dp,
00563   double p);
00564
00566 double dx2deg(
00567   double dx,
00568   double lat);
00569
00571 double dy2deg(
00572   double dy);
00573
00575 double dz2dp(
00576   double dz,
00577   double p);
00578
00580 void geo2cart(
00581   double z,
00582   double lon,
00583   double lat,
00584   double *x);
00585
00587 void get_met(
00588   ctl_t * ctl,
00589   char *metbase,
00590   double t,
00591   met_t * met0,
00592   met_t * met1);
00593
00595 void get_met_help(
00596   double t,
00597   int direct,
00598   char *metbase,
00599   double dt_met,
00600   char *filename);
00601
00603 void intpol_met_2d(
00604   double array[EX][EY],
00605   int ix,
00606   int iy,
00607   double wx,
00608   double wy,
00609   double *var);
00610
00612 void intpol_met_3d(
00613   float array[EX][EY][EP],
00614   int ip,
00615   int ix,
00616   int iy,
00617   double wp,
```

```
00618   double wx,
00619   double wy,
00620   double *var);
00621
00623 void intpol_met_space(
00624   met_t * met,
00625   double p,
00626   double lon,
00627   double lat,
00628   double *ps,
00629   double *t,
00630   double *u,
00631   double *v,
00632   double *w,
00633   double *h2o,
00634   double *o3);
00635
00637 void intpol_met_time(
00638   met_t * met0,
00639   met_t * met1,
00640   double ts,
00641   double p,
00642   double lon,
00643   double lat,
00644   double *ps,
00645   double *t,
00646   double *u,
00647   double *v,
00648   double *w,
00649   double *h2o,
00650   double *o3);
00651
00653 void jsec2time(
00654   double jsec,
00655   int *year,
00656   int *mon,
00657   int *day,
00658   int *hour,
00659   int *min,
00660   int *sec,
00661   double *remain);
00662
00664 int locate(
00665   double *xx,
00666   int n,
00667   double x);
00668
00670 void read_atm(
00671   const char *filename,
00672   ctl_t * ctl,
00673   atm_t * atm);
00674
00676 void read_ctl(
00677   const char *filename,
00678   int argc,
00679   char *argv[],
00680   ctl_t * ctl);
00681
00683 void read_met(
00684   ctl_t * ctl,
00685   char *filename,
00686   met_t * met);
00687
00689 void read_met_extrapolate(
00690   met_t * met);
00691
00693 void read_met_help(
00694   int ncid,
00695   char *varname,
00696   char *varname2,
00697   met_t * met,
00698   float dest[EX][EY][EP],
00699   float scl);
00700
00702 void read_met_ml2pl(
00703   ctl_t * ctl,
00704   met_t * met,
00705   float var[EX][EY][EP]);
00706
00708 void read_met_periodic(
00709   met_t * met);
00710
00712 double scan_ctl(
00713   const char *filename,
00714   int argc,
00715   char *argv[],
00716   const char *varname,
```

```
00717   int arridx,
00718   const char *defvalue,
00719   char *value);
00720
00722 void time2jsec(
00723   int year,
00724   int mon,
00725   int day,
00726   int hour,
00727   int min,
00728   int sec,
00729   double remain,
00730   double *jsec);
00731
00733 void timer(
00734   const char *name,
00735   int id,
00736   int mode);
00737
00738 /* Get tropopause pressure... */
00739 double tropopause(
00740   double t,
00741   double lat);
00742
00744 void write_atm(
00745   const char *filename,
00746   ctl_t * ctl,
00747   atm_t * atm,
00748   double t);
00749
00751 void write_csi(
00752   const char *filename,
00753   ctl_t * ctl,
00754   atm_t * atm,
00755   double t);
00756
00758 void write_ens(
00759   const char *filename,
00760   ctl_t * ctl,
00761   atm_t * atm,
00762   double t);
00763
00765 void write_grid(
00766   const char *filename,
00767   ctl_t * ctl,
00768   met_t * met0,
00769   met_t * met1,
00770   atm_t * atm,
00771   double t);
00772
00774 void write_prof(
00775   const char *filename,
00776   ctl_t * ctl,
00777   met_t * met0,
00778   met_t * met1,
00779   atm_t * atm,
00780   double t);
00781
00783 void write_station(
00784   const char *filename,
00785   ctl_t * ctl,
00786   atm_t * atm,
00787   double t);
```

5.15 match.c File Reference

Calculate deviations between two trajectories.

Functions

- int main (int argc, char ∗argv[ ])

5.15.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

5.15.2 Function Documentation

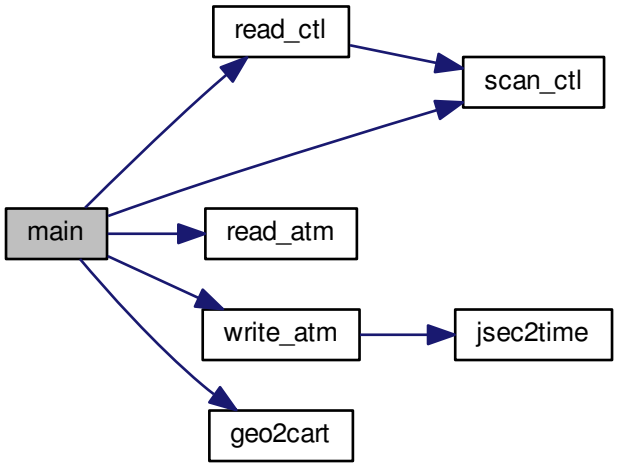
5.15.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 28 of file match.c.

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

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

Here is the call graph for this function:



5.16 match.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029   int argc,
00030   char *argv[]) {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm1, *atm2, *atm3;
00035
00036   FILE *out;
00037
00038   char filename[LEN];
00039
00040   double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00041
00042   int filter, ip1, ip2, iq, n;
00043
00044   /* Allocate... */
00045   ALLOC(atm1, atm_t, 1);
00046   ALLOC(atm2, atm_t, 1);
00047   ALLOC(atm3, atm_t, 1);
00048
00049   /* Check arguments... */
```

```
00050   if (argc < 5)
00051     ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00052
00053   /* Read control parameters... */
00054   read_ctl(argv[1], argc, argv, &ctl);
00055   filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00056   filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00057
00058   /* Read atmospheric data... */
00059   read_atm(argv[2], &ctl, atm1);
00060   read_atm(argv[3], &ctl, atm2);
00061
00062   /* Write info... */
00063   printf("Write transport deviations: %s\n", argv[4]);
00064
00065   /* Create output file... */
00066   if (!(out = fopen(argv[4], "w")))
00067     ERRMSG("Cannot create file!");
00068
00069   /* Write header... */
00070   fprintf(out,
00071           "# $1 = time [s]\n"
00072           "# $2 = altitude [km]\n"
00073           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00074   for (iq = 0; iq < ctl.nq; iq++)
00075     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00076             ctl.qnt_unit[iq]);
00077   fprintf(out,
00078           "# $%d = trajectory time [s]\n"
00079           "# $%d = vertical length of trajectory [km]\n"
00080           "# $%d = horizontal length of trajectory [km]\n"
00081           "# $%d = vertical deviation [km]\n"
00082           "# $%d = horizontal deviation [km]\n",
00083           5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
00084   for (iq = 0; iq < ctl.nq; iq++)
00085     fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00086             ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00087   fprintf(out, "\n");
00088
00089   /* Filtering of reference time series... */
00090   if (filter) {
00091
00092     /* Copy data... */
00093     memcpy(atm3, atm2, sizeof(atm_t));
00094
00095     /* Loop over data points... */
00096     for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097       n = 0;
00098       atm2->p[ip1] = 0;
00099       for (iq = 0; iq < ctl.nq; iq++)
00100         atm2->q[iq][ip1] = 0;
00101       for (ip2 = 0; ip2 < atm2->np; ip2++)
00102         if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
00103           atm2->p[ip1] += atm3->p[ip2];
00104           for (iq = 0; iq < ctl.nq; iq++)
00105             atm2->q[iq][ip1] += atm3->q[iq][ip2];
00106           n++;
00107         }
00108       atm2->p[ip1] /= n;
00109       for (iq = 0; iq < ctl.nq; iq++)
00110         atm2->q[iq][ip1] /= n;
00111     }
00112
00113     /* Write filtered data... */
00114     sprintf(filename, "%s.filt", argv[3]);
00115     write_atm(filename, &ctl, atm2, 0);
00116   }
00117
00118   /* Loop over air parcels (reference data)... */
00119   for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121     /* Get trajectory length... */
00122     if (ip2 > 0) {
00123       geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00124       geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00125       lh += DIST(x1, x2);
00126       lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00127       lt = fabs(atm2->time[ip2] - atm2->time[0]);
00128     }
00129
00130     /* Init... */
00131     n = 0;
00132     dh = 0;
00133     dv = 0;
00134     for (iq = 0; iq < ctl.nq; iq++)
00135       dq[iq] = 0;
00136     geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
```

```
00137
00138     /* Find corresponding time step (test data)... */
00139     for (ip1 = 0; ip1 < atm1->np; ip1++)
00140       if (fabs(atm1->time[ip1] - atm2->time[ip2])
00141           < (filter ? filter_dt : 0.1)) {
00142
00143         /* Calculate deviations... */
00144         geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00145         dh += DIST(x1, x2);
00146         dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00147         for (iq = 0; iq < ctl.nq; iq++)
00148           dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00149         n++;
00150       }
00151
00152     /* Write output... */
00153     if (n > 0) {
00154       fprintf(out, "%.2f %.4f %.4f %.4f",
00155               atm2->time[ip2], Z(atm2->p[ip2]),
00156               atm2->lon[ip2], atm2->lat[ip2]);
00157       for (iq = 0; iq < ctl.nq; iq++) {
00158         fprintf(out, " ");
00159         fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00160       }
00161       fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00162       for (iq = 0; iq < ctl.nq; iq++) {
00163         fprintf(out, " ");
00164         fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
00165       }
00166       fprintf(out, "\n");
00167     }
00168   }
00169
00170   /* Close file... */
00171   fclose(out);
00172
00173   /* Free... */
00174   free(atm1);
00175   free(atm2);
00176   free(atm3);
00177
00178   return EXIT_SUCCESS;
00179 }
```

5.17 met_map.c File Reference

Extract global map from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.17.1 Detailed Description

Extract global map from meteorological data.

Definition in file met_map.c.

5.17.2 Function Documentation

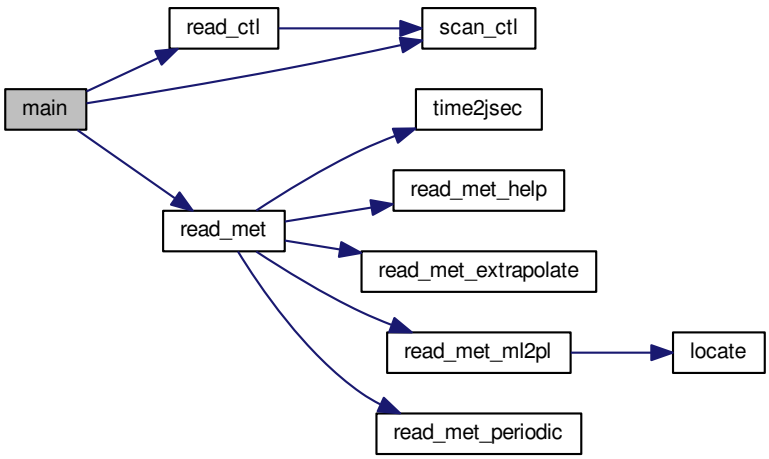
5.17.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 27 of file met_map.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00038     um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00039
00040   static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042   /* Allocate... */
00043   ALLOC(met, met_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 4)
00047     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00052
00053   /* Loop over files... */
00054   for (i = 3; i < argc; i++) {
00055
00056     /* Read meteorological data... */
00057     if (!(in = fopen(argv[i], "r")))
00058       continue;
00059     else
00060       fclose(in);
00061     read_met(&ctl, argv[i], met);
00062
00063     /* Find nearest pressure level... */
00064     for (ip2 = 0; ip2 < met->np; ip2++) {
00065       dz = fabs(Z(met->p[ip2]) - z);
00066       if (dz < dzmin) {
00067         dzmin = dz;
00068         ip = ip2;
00069       }
00070     }
00071
00072     /* Average data... */
00073     for (ix = 0; ix < met->nx; ix++)
00074       for (iy = 0; iy < met->ny; iy++) {
00075         timem[ix][iy] += met->time;
00076         tm[ix][iy] += met->t[ix][iy][ip];
00077         um[ix][iy] += met->u[ix][iy][ip];
00078         vm[ix][iy] += met->v[ix][iy][ip];
00079         wm[ix][iy] += met->w[ix][iy][ip];
00080         h2om[ix][iy] += met->h2o[ix][iy][ip];
00081         o3m[ix][iy] += met->o3[ix][iy][ip];
00082         psm[ix][iy] += met->ps[ix][iy];
00083         np[ix][iy]++;
00084       }
00085   }
00086
00087   /* Create output file... */
00088   printf("Write meteorological data file: %s\n", argv[2]);
00089   if (!(out = fopen(argv[2], "w")))
00090     ERRMSG("Cannot create file!");
00091
00092   /* Write header... */
00093   fprintf(out,
00094           "# $1  = time [s]\n"
00095           "# $2  = altitude [km]\n"
00096           "# $3  = longitude [deg]\n"
00097           "# $4  = latitude [deg]\n"
00098           "# $5  = pressure [hPa]\n"
00099           "# $6  = temperature [K]\n"
00100           "# $7  = zonal wind [m/s]\n"
00101           "# $8  = meridional wind [m/s]\n"
00102           "# $9  = vertical wind [hPa/s]\n"
00103           "# $10 = H2O volume mixing ratio [1]\n"
00104           "# $11 = O3 volume mixing ratio [1]\n"
00105           "# $12 = surface pressure [hPa]\n");
00106
00107   /* Write data... */
00108   for (iy = 0; iy < met->ny; iy++) {
00109     fprintf(out, "\n");
00110     for (ix = 0; ix < met->nx; ix++)
00111       if (met->lon[ix] >= 180)
00112         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00113                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00114                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00115                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
```

```
00116                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00117                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00118                 psm[ix][iy] / np[ix][iy]);
00119     for (ix = 0; ix < met->nx; ix++)
00120       if (met->lon[ix] <= 180)
00121         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00122                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00123                 met->lon[ix], met->lat[iy], met->p[ip],
00124                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00125                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00126                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00127                 psm[ix][iy] / np[ix][iy]);
00128   }
00129
00130   /* Close file... */
00131   fclose(out);
00132
00133   /* Free... */
00134   free(met);
00135
00136   return EXIT_SUCCESS;
00137 }
```

Here is the call graph for this function:



5.18 met_map.c

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

```
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00038     um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00039
00040   static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042   /* Allocate... */
00043   ALLOC(met, met_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 4)
00047     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00052
00053   /* Loop over files... */
00054   for (i = 3; i < argc; i++) {
00055
00056     /* Read meteorological data... */
00057     if (!(in = fopen(argv[i], "r")))
00058       continue;
00059     else
00060       fclose(in);
00061     read_met(&ctl, argv[i], met);
00062
00063     /* Find nearest pressure level... */
00064     for (ip2 = 0; ip2 < met->np; ip2++) {
00065       dz = fabs(Z(met->p[ip2]) - z);
00066       if (dz < dzmin) {
00067         dzmin = dz;
00068         ip = ip2;
00069       }
00070     }
00071
00072     /* Average data... */
00073     for (ix = 0; ix < met->nx; ix++)
00074       for (iy = 0; iy < met->ny; iy++) {
00075         timem[ix][iy] += met->time;
00076         tm[ix][iy] += met->t[ix][iy][ip];
00077         um[ix][iy] += met->u[ix][iy][ip];
00078         vm[ix][iy] += met->v[ix][iy][ip];
00079         wm[ix][iy] += met->w[ix][iy][ip];
00080         h2om[ix][iy] += met->h2o[ix][iy][ip];
00081         o3m[ix][iy] += met->o3[ix][iy][ip];
00082         psm[ix][iy] += met->ps[ix][iy];
00083         np[ix][iy]++;
00084       }
00085   }
00086
00087   /* Create output file... */
00088   printf("Write meteorological data file: %s\n", argv[2]);
00089   if (!(out = fopen(argv[2], "w")))
00090     ERRMSG("Cannot create file!");
00091
00092   /* Write header... */
00093   fprintf(out,
00094           "# $1  = time [s]\n"
00095           "# $2  = altitude [km]\n"
00096           "# $3  = longitude [deg]\n"
00097           "# $4  = latitude [deg]\n"
00098           "# $5  = pressure [hPa]\n"
00099           "# $6  = temperature [K]\n"
00100           "# $7  = zonal wind [m/s]\n"
00101           "# $8  = meridional wind [m/s]\n"
00102           "# $9  = vertical wind [hPa/s]\n"
00103           "# $10 = H2O volume mixing ratio [1]\n"
00104           "# $11 = O3 volume mixing ratio [1]\n"
00105           "# $12 = surface pressure [hPa]\n");
00106
00107   /* Write data... */
00108   for (iy = 0; iy < met->ny; iy++) {
00109     fprintf(out, "\n");
00110     for (ix = 0; ix < met->nx; ix++)
00111       if (met->lon[ix] >= 180)
00112         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00113                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
```

```
00114                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00115                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00116                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00117                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00118                 psm[ix][iy] / np[ix][iy]);
00119     for (ix = 0; ix < met->nx; ix++)
00120       if (met->lon[ix] <= 180)
00121         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00122                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00123                 met->lon[ix], met->lat[iy], met->p[ip],
00124                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00125                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00126                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00127                 psm[ix][iy] / np[ix][iy]);
00128   }
00129
00130   /* Close file... */
00131   fclose(out);
00132
00133   /* Free... */
00134   free(met);
00135
00136   return EXIT_SUCCESS;
00137 }
```

5.19 met_prof.c File Reference

Extract vertical profile from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.19.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met_prof.c.

5.19.2 Function Documentation

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

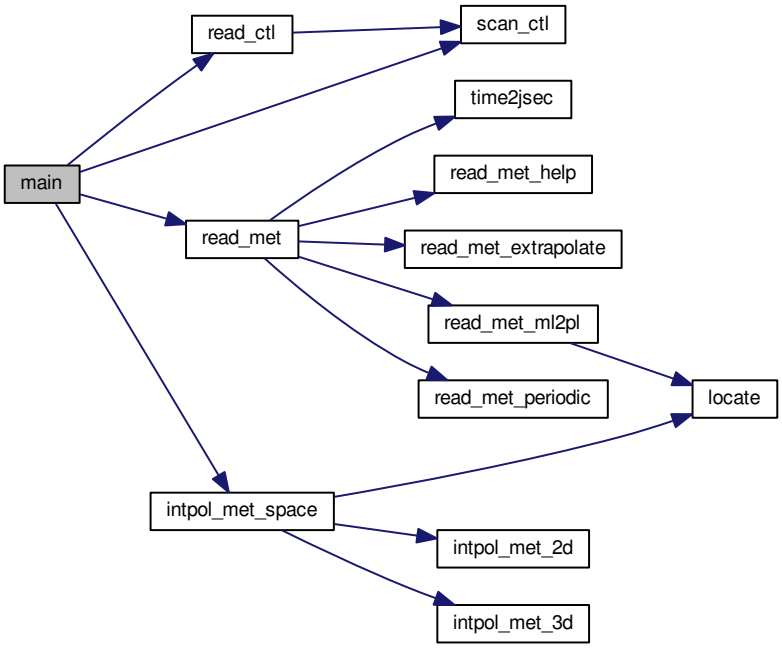
Definition at line 38 of file met_prof.c.

```
00040                 {
00041
00042   ctl_t ctl;
00043
00044   met_t *met;
00045
00046   FILE *in, *out;
00047
00048   static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ],
00050     w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00051
00052   static int i, iz, np[NZ];
00053
00054   /* Allocate... */
00055   ALLOC(met, met_t, 1);
00056
00057   /* Check arguments... */
00058   if (argc < 4)
00059     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
```

```
00060
00061   /* Read control parameters... */
00062   read_ctl(argv[1], argc, argv, &ctl);
00063   z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
00064   z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
00065   dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
00066   lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00067   lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00068   dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
00069   lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
00070   lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00071   dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00072
00073   /* Loop over input files... */
00074   for (i = 3; i < argc; i++) {
00075
00076     /* Read meteorological data... */
00077     if (!(in = fopen(argv[i], "r")))
00078       continue;
00079     else
00080       fclose(in);
00081     read_met(&ctl, argv[i], met);
00082
00083     /* Average... */
00084     for (z = z0; z <= z1; z += dz) {
00085       iz = (int) ((z - z0) / dz);
00086       if (iz < 0 || iz > NZ)
00087         ERRMSG("Too many altitudes!");
00088       for (lon = lon0; lon <= lon1; lon += dlon)
00089         for (lat = lat0; lat <= lat1; lat += dlat) {
00090           intpol_met_space(met, P(z), lon, lat, &ps,
00091                            &t, &u, &v, &w, &h2o, &o3);
00092           if (gsl_finite(t) && gsl_finite(u)
00093               && gsl_finite(v) && gsl_finite(w)) {
00094             timem[iz] += met->time;
00095             lonm[iz] += lon;
00096             latm[iz] += lat;
00097             tm[iz] += t;
00098             um[iz] += u;
00099             vm[iz] += v;
00100             wm[iz] += w;
00101             h2om[iz] += h2o;
00102             o3m[iz] += o3;
00103             psm[iz] += ps;
00104             np[iz]++;
00105           }
00106         }
00107     }
00108   }
00109
00110   /* Normalize... */
00111   for (z = z0; z <= z1; z += dz) {
00112     iz = (int) ((z - z0) / dz);
00113     if (np[iz] > 0) {
00114       timem[iz] /= np[iz];
00115       lonm[iz] /= np[iz];
00116       latm[iz] /= np[iz];
00117       tm[iz] /= np[iz];
00118       um[iz] /= np[iz];
00119       vm[iz] /= np[iz];
00120       wm[iz] /= np[iz];
00121       h2om[iz] /= np[iz];
00122       o3m[iz] /= np[iz];
00123       psm[iz] /= np[iz];
00124     } else {
00125       timem[iz] = GSL_NAN;
00126       lonm[iz] = GSL_NAN;
00127       latm[iz] = GSL_NAN;
00128       tm[iz] = GSL_NAN;
00129       um[iz] = GSL_NAN;
00130       vm[iz] = GSL_NAN;
00131       wm[iz] = GSL_NAN;
00132       h2om[iz] = GSL_NAN;
00133       o3m[iz] = GSL_NAN;
00134       psm[iz] = GSL_NAN;
00135     }
00136   }
00137
00138   /* Create output file... */
00139   printf("Write meteorological data file: %s\n", argv[2]);
00140   if (!(out = fopen(argv[2], "w")))
00141     ERRMSG("Cannot create file!");
00142
00143   /* Write header... */
00144   fprintf(out,
00145           "# $1  = time [s]\n"
00146           "# $2  = altitude [km]\n"
```

```
00147           "# $3  = longitude [deg]\n"
00148           "# $4  = latitude [deg]\n"
00149           "# $5  = pressure [hPa]\n"
00150           "# $6  = temperature [K]\n"
00151           "# $7  = zonal wind [m/s]\n"
00152           "# $8  = meridional wind [m/s]\n"
00153           "# $9  = vertical wind [hPa/s]\n"
00154           "# $10 = H2O volume mixing ratio [1]\n"
00155           "# $11 = O3 volume mixing ratio [1]\n"
00156           "# $12 = surface pressure [hPa]\n\n");
00157
00158   /* Write data... */
00159   for (z = z0; z <= z1; z += dz) {
00160     iz = (int) ((z - z0) / dz);
00161     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00162             timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz],
00163             um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz], psm[iz]);
00164   }
00165
00166   /* Close file... */
00167   fclose(out);
00168
00169   /* Free... */
00170   free(met);
00171
00172   return EXIT_SUCCESS;
00173 }
```

Here is the call graph for this function:



5.20 met_prof.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
```

```
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* ------------------------------------------------------------
00028    Dimensions...
00029    ------------------------------------------------------------ */
00030
00031 /* Maximum number of altitudes. */
00032 #define NZ 1000
00033
00034 /* ------------------------------------------------------------
00035    Main...
00036    ------------------------------------------------------------ */
00037
00038 int main(
00039   int argc,
00040   char *argv[]) {
00041
00042   ctl_t ctl;
00043
00044   met_t *met;
00045
00046   FILE *in, *out;
00047
00048   static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ],
00050     w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00051
00052   static int i, iz, np[NZ];
00053
00054   /* Allocate... */
00055   ALLOC(met, met_t, 1);
00056
00057   /* Check arguments... */
00058   if (argc < 4)
00059     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00060
00061   /* Read control parameters... */
00062   read_ctl(argv[1], argc, argv, &ctl);
00063   z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
00064   z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
00065   dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
00066   lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00067   lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00068   dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
00069   lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
00070   lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00071   dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00072
00073   /* Loop over input files... */
00074   for (i = 3; i < argc; i++) {
00075
00076     /* Read meteorological data... */
00077     if (!(in = fopen(argv[i], "r")))
00078       continue;
00079     else
00080       fclose(in);
00081     read_met(&ctl, argv[i], met);
00082
00083     /* Average... */
00084     for (z = z0; z <= z1; z += dz) {
00085       iz = (int) ((z - z0) / dz);
00086       if (iz < 0 || iz > NZ)
00087         ERRMSG("Too many altitudes!");
00088       for (lon = lon0; lon <= lon1; lon += dlon)
00089         for (lat = lat0; lat <= lat1; lat += dlat) {
00090           intpol_met_space(met, P(z), lon, lat, &ps,
00091                            &t, &u, &v, &w, &h2o, &o3);
00092           if (gsl_finite(t) && gsl_finite(u)
00093               && gsl_finite(v) && gsl_finite(w)) {
00094             timem[iz] += met->time;
00095             lonm[iz] += lon;
00096             latm[iz] += lat;
00097             tm[iz] += t;
00098             um[iz] += u;
00099             vm[iz] += v;
00100             wm[iz] += w;
```

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

5.21 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

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

5.21.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

5.21.2 Function Documentation

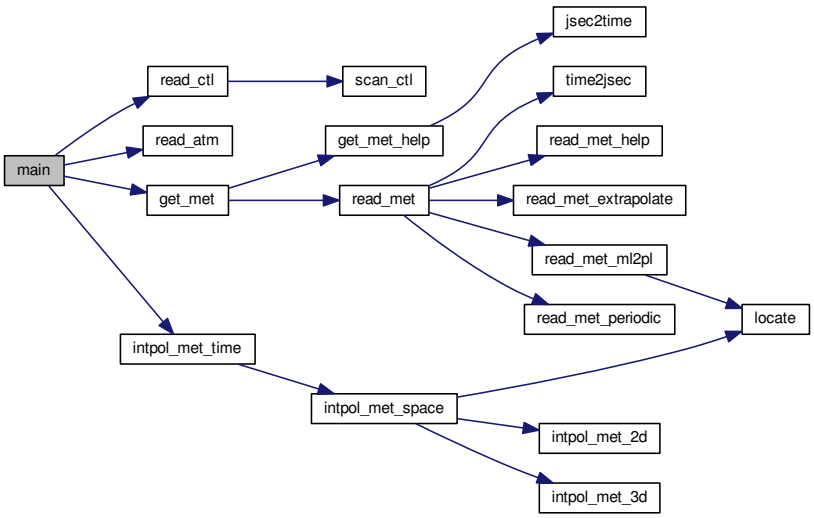
5.21.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 31 of file met_sample.c.

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

```
00097   /* Close file... */
00098   fclose(out);
00099
00100   /* Free... */
00101   free(atm);
00102   free(met0);
00103   free(met1);
00104
00105   return EXIT_SUCCESS;
00106 }
```

Here is the call graph for this function:



5.22 met_sample.c

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

5.23 met_zm.c File Reference

Extract zonal mean from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.23.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met_zm.c.

5.23.2 Function Documentation

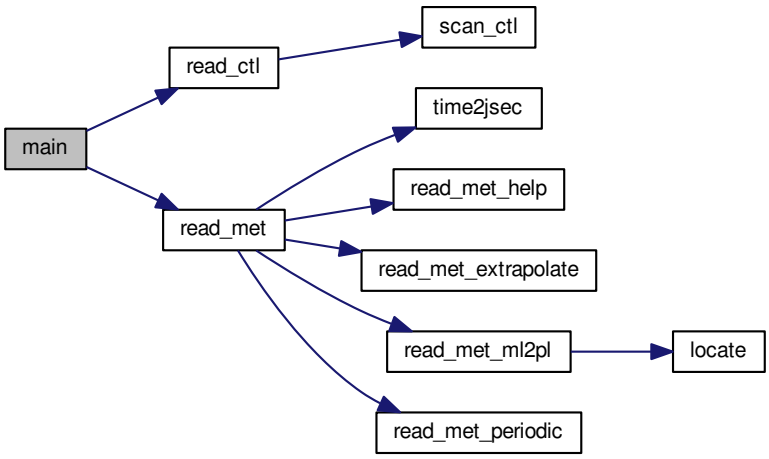
5.23.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file met_zm.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
00038     vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],
00039     psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vhm2[EP][EY],
00040     wm2[EP][EY], h2om2[EP][EY], o3m2[EP][EY];
00041
00042   static int i, ip, ix, iy, np[EP][EY];
00043
00044   /* Allocate... */
00045   ALLOC(met, met_t, 1);
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00050
00051   /* Read control parameters... */
00052   read_ctl(argv[1], argc, argv, &ctl);
00053
00054   /* Loop over files... */
00055   for (i = 3; i < argc; i++) {
00056
00057     /* Read meteorological data... */
00058     if (!(in = fopen(argv[i], "r")))
00059       continue;
00060     else
00061       fclose(in);
00062     read_met(&ctl, argv[i], met);
00063
00064     /* Average data... */
00065     for (ix = 0; ix < met->nx; ix++)
00066       for (iy = 0; iy < met->ny; iy++)
00067         for (ip = 0; ip < met->np; ip++) {
00068           timem[ip][iy] += met->time;
00069           tm[ip][iy] += met->t[ix][iy][ip];
00070           um[ip][iy] += met->u[ix][iy][ip];
00071           vm[ip][iy] += met->v[ix][iy][ip];
00072           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00073                               + gsl_pow_2(met->v[ix][iy][ip]));
00074           wm[ip][iy] += met->w[ix][iy][ip];
00075           h2om[ip][iy] += met->h2o[ix][iy][ip];
00076           o3m[ip][iy] += met->o3[ix][iy][ip];
00077           psm[ip][iy] += met->ps[ix][iy];
00078           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00079           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00080           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00081           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00082             + gsl_pow_2(met->v[ix][iy][ip]);
00083           wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00084           h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00085           o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00086           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00087           np[ip][iy]++;
00088         }
00089   }
00090
00091   /* Create output file... */
00092   printf("Write meteorological data file: %s\n", argv[2]);
00093   if (!(out = fopen(argv[2], "w")))
```

```
00094     ERRMSG("Cannot create file!");
00095
00096   /* Write header... */
00097   fprintf(out,
00098           "# $1  = time [s]\n"
00099           "# $2  = altitude [km]\n"
00100           "# $3  = latitude [deg]\n"
00101           "# $4  = temperature mean [K]\n"
00102           "# $5  = temperature standard deviation [K]\n"
00103           "# $6  = zonal wind mean [m/s]\n"
00104           "# $7  = zonal wind standard deviation [m/s]\n"
00105           "# $8  = meridional wind mean [m/s]\n"
00106           "# $9  = meridional wind standard deviation [m/s]\n"
00107           "# $10 = horizontal wind mean [m/s]\n"
00108           "# $11 = horizontal wind standard deviation [m/s]\n"
00109           "# $12 = vertical wind mean [hPa/s]\n"
00110           "# $13 = vertical wind standard deviation [hPa/s]\n"
00111           "# $14 = H2O vmr mean [1]\n"
00112           "# $15 = H2O vmr standard deviation [1]\n"
00113           "# $16 = O3 vmr mean [1]\n"
00114           "# $17 = O3 vmr standard deviation [1]\n"
00115           "# $18 = surface pressure mean [hPa]\n"
00116           "# $19 = surface pressure standard deviation [hPa]\n");
00117
00118   /* Write data... */
00119   for (iy = 0; iy < met->ny; iy++) {
00120     fprintf(out, "\n");
00121     for (ip = 0; ip < met->np; ip++)
00122       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00123               " %g %g %g %g %g %g %g %g\n",
00124               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
00125               tm[ip][iy] / np[ip][iy],
00126               sqrt(tm2[ip][iy] / np[ip][iy] -
00127                    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00128               um[ip][iy] / np[ip][iy],
00129               sqrt(um2[ip][iy] / np[ip][iy] -
00130                    gsl_pow_2(um[ip][iy] / np[ip][iy])),
00131               vm[ip][iy] / np[ip][iy],
00132               sqrt(vm2[ip][iy] / np[ip][iy] -
00133                    gsl_pow_2(vm[ip][iy] / np[ip][iy])),
00134               vhm[ip][iy] / np[ip][iy],
00135               sqrt(vhm2[ip][iy] / np[ip][iy] -
00136                    gsl_pow_2(vhm[ip][iy] / np[ip][iy])),
00137               wm[ip][iy] / np[ip][iy],
00138               sqrt(wm2[ip][iy] / np[ip][iy] -
00139                    gsl_pow_2(wm[ip][iy] / np[ip][iy])),
00140               h2om[ip][iy] / np[ip][iy],
00141               sqrt(h2om2[ip][iy] / np[ip][iy] -
00142                    gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
00143               o3m[ip][iy] / np[ip][iy],
00144               sqrt(o3m2[ip][iy] / np[ip][iy] -
00145                    gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
00146               psm[ip][iy] / np[ip][iy],
00147               sqrt(psm2[ip][iy] / np[ip][iy] -
00148                    gsl_pow_2(psm[ip][iy] / np[ip][iy])));
00149   }
00150
00151   /* Close file... */
00152   fclose(out);
00153
00154   /* Free... */
00155   free(met);
00156
00157   return EXIT_SUCCESS;
00158 }
```

Here is the call graph for this function:



5.24 met_zm.c

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

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

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

5.25 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

Functions

- int main (int argc, char ∗argv[ ])

5.25.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

5.25.2 Function Documentation

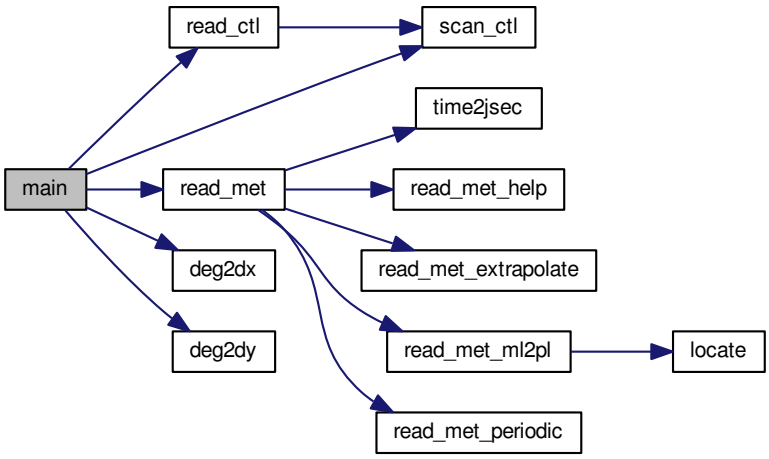
5.25.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 8 of file smago.c.

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

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

Here is the call graph for this function:



5.26 smago.c

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

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

5.27 split.c File Reference

Split air parcels into a larger number of parcels.

Functions

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

5.27.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

5.27.2 Function Documentation

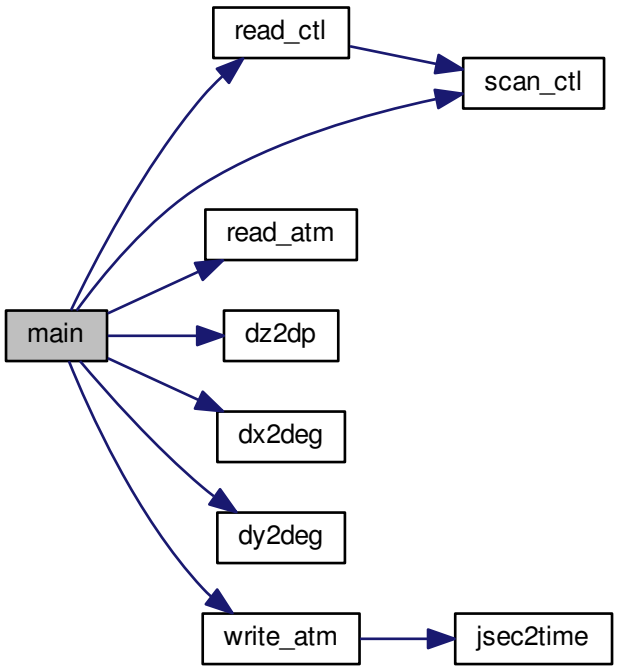
5.27.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file split.c.

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

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

Here is the call graph for this function:



5.28 split.c

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

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

5.29 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

- int main (int argc, char ∗argv[ ])

5.29.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

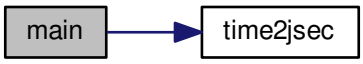
5.29.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

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

Here is the call graph for this function:



5.30 time2jsec.c

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

5.31 trac.c File Reference

Lagrangian particle dispersion model.

Functions

- void init_simtime (ctl_t *ctl, atm_t *atm)
  Set simulation time interval.
- void module_advection (met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)
  Calculate advection of air parcels.
- void module_decay (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)
  Calculate exponential decay of particle mass.
- void module_diffusion_meso (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt, gsl_rng *rng)
  Calculate mesoscale diffusion.
- void module_diffusion_turb (ctl_t *ctl, atm_t *atm, int ip, double dt, gsl_rng *rng)
  Calculate turbulent diffusion.
- void module_isosurf (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip)
  Force air parcels to stay on isosurface.
- void module_meteo (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip)
  Interpolate meteorological data for air parcel positions.
- 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[])

5.31.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.31.2 Function Documentation

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

Set simulation time interval.

Definition at line 408 of file trac.c.

```
00410                {
00411
00412   /* Set inital and final time... */
00413   if (ctl->direction == 1) {
00414     if (ctl->t_start < -1e99)
00415       ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00416     if (ctl->t_stop < -1e99)
00417       ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00418   } else if (ctl->direction == -1) {
00419     if (ctl->t_stop < -1e99)
00420       ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00421     if (ctl->t_start < -1e99)
00422       ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00423   }
00424
00425   /* Check time... */
00426   if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
00427     ERRMSG("Nothing to do!");
00428 }
```

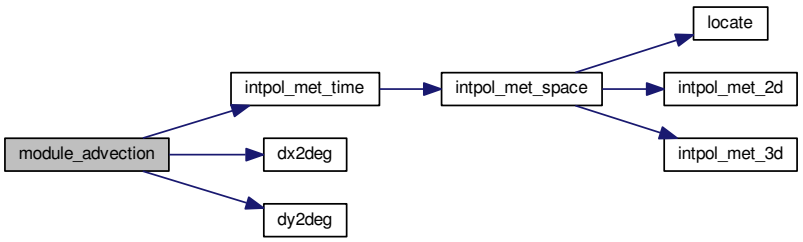
5.31.2.2 void module_advection (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate advection of air parcels.

Definition at line 432 of file trac.c.

```
00437              {
00438
00439   double v[3], xm[3];
00440
00441   /* Interpolate meteorological data... */
00442   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00443                   atm->lon[ip], atm->lat[ip], NULL, NULL,
00444                   &v[0], &v[1], &v[2], NULL, NULL);
00445
00446   /* Get position of the mid point... */
00447   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00448   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00449   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00450
00451   /* Interpolate meteorological data for mid point... */
00452   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00453                   xm[2], xm[0], xm[1], NULL, NULL,
00454                   &v[0], &v[1], &v[2], NULL, NULL);
00455
00456   /* Save new position... */
00457   atm->time[ip] += dt;
00458   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00459   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00460   atm->p[ip] += dt * v[2];
00461 }
```

Here is the call graph for this function:



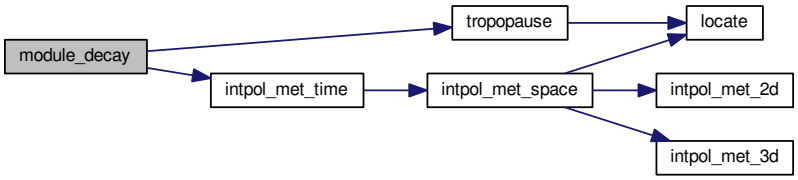
5.31.2.3 void module_decay (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate exponential decay of particle mass.

Definition at line 465 of file trac.c.

```
00471             {
00472
00473   double ps, pt, tdec;
00474
00475   /* Check lifetime values... */
00476   if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00477     return;
00478
00479   /* Set constant lifetime... */
00480   if (ctl->tdec_trop == ctl->tdec_strat)
00481     tdec = ctl->tdec_trop;
00482
00483   /* Set altitude-dependent lifetime... */
00484   else {
00485
00486     /* Get surface pressure... */
00487     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00488                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00489                     NULL, NULL, NULL, NULL, NULL);
00490
00491     /* Get tropopause pressure... */
00492     pt = tropopause(atm->time[ip], atm->lat[ip]);
00493
00494     /* Set lifetime... */
00495     if (atm->p[ip] <= pt)
00496       tdec = ctl->tdec_strat;
00497     else
00498       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00499   }
00500
00501   /* Calculate exponential decay... */
00502   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00503 }
```

Here is the call graph for this function:



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

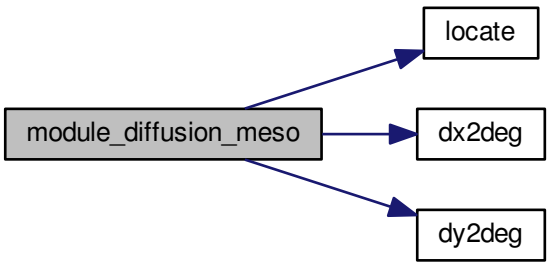
Calculate mesoscale diffusion.

Definition at line 507 of file trac.c.

```
00514                 {
00515
00516   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00517
00518   int ix, iy, iz;
00519
00520   /* Calculate mesoscale velocity fluctuations... */
00521   if (ctl->turb_meso > 0) {
00522
00523     /* Get indices... */
00524     ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00525     iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00526     iz = locate(met0->p, met0->np, atm->p[ip]);
00527
00528     /* Collect local wind data... */
00529     u[0] = met0->u[ix][iy][iz];
00530     u[1] = met0->u[ix + 1][iy][iz];
00531     u[2] = met0->u[ix][iy + 1][iz];
00532     u[3] = met0->u[ix + 1][iy + 1][iz];
00533     u[4] = met0->u[ix][iy][iz + 1];
00534     u[5] = met0->u[ix + 1][iy][iz + 1];
00535     u[6] = met0->u[ix][iy + 1][iz + 1];
00536     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00537
00538     v[0] = met0->v[ix][iy][iz];
00539     v[1] = met0->v[ix + 1][iy][iz];
00540     v[2] = met0->v[ix][iy + 1][iz];
00541     v[3] = met0->v[ix + 1][iy + 1][iz];
00542     v[4] = met0->v[ix][iy][iz + 1];
00543     v[5] = met0->v[ix + 1][iy][iz + 1];
00544     v[6] = met0->v[ix][iy + 1][iz + 1];
00545     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00546
00547     w[0] = met0->w[ix][iy][iz];
00548     w[1] = met0->w[ix + 1][iy][iz];
00549     w[2] = met0->w[ix][iy + 1][iz];
00550     w[3] = met0->w[ix + 1][iy + 1][iz];
00551     w[4] = met0->w[ix][iy][iz + 1];
00552     w[5] = met0->w[ix + 1][iy][iz + 1];
00553     w[6] = met0->w[ix][iy + 1][iz + 1];
00554     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00555
00556     /* Get indices... */
00557     ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00558     iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00559     iz = locate(met1->p, met1->np, atm->p[ip]);
00560
00561     /* Collect local wind data... */
00562     u[8] = met1->u[ix][iy][iz];
00563     u[9] = met1->u[ix + 1][iy][iz];
00564     u[10] = met1->u[ix][iy + 1][iz];
00565     u[11] = met1->u[ix + 1][iy + 1][iz];
00566     u[12] = met1->u[ix][iy][iz + 1];
00567     u[13] = met1->u[ix + 1][iy][iz + 1];
00568     u[14] = met1->u[ix][iy + 1][iz + 1];
00569     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00570
00571     v[8] = met1->v[ix][iy][iz];
00572     v[9] = met1->v[ix + 1][iy][iz];
00573     v[10] = met1->v[ix][iy + 1][iz];
00574     v[11] = met1->v[ix + 1][iy + 1][iz];
00575     v[12] = met1->v[ix][iy][iz + 1];
00576     v[13] = met1->v[ix + 1][iy][iz + 1];
00577     v[14] = met1->v[ix][iy + 1][iz + 1];
00578     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00579
00580     w[8] = met1->w[ix][iy][iz];
00581     w[9] = met1->w[ix + 1][iy][iz];
00582     w[10] = met1->w[ix][iy + 1][iz];
00583     w[11] = met1->w[ix + 1][iy + 1][iz];
00584     w[12] = met1->w[ix][iy][iz + 1];
00585     w[13] = met1->w[ix + 1][iy][iz + 1];
00586     w[14] = met1->w[ix][iy + 1][iz + 1];
00587     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00588
00589     /* Get standard deviations of local wind data... */
00590     usig = gsl_stats_sd(u, 1, 16);
00591     vsig = gsl_stats_sd(v, 1, 16);
00592     wsig = gsl_stats_sd(w, 1, 16);
00593
00594     /* Set temporal correlations for mesoscale fluctuations... */
00595     r = 1 - 2 * fabs(dt) / ctl->dt_met;
00596     rs = sqrt(1 - r * r);
00597
00598     /* Calculate mesoscale wind fluctuations... */
00599     atm->up[ip] =
00600       r * atm->up[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
```

```
00601                                                   ctl->turb_meso * usig);
00602     atm->vp[ip] =
00603       r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00604                                                   ctl->turb_meso * vsig);
00605     atm->wp[ip] =
00606       r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00607                                                   ctl->turb_meso * wsig);
00608
00609     /* Calculate air parcel displacement... */
00610     atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00611     atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00612     atm->p[ip] += atm->wp[ip] * dt;
00613   }
00614 }
```

Here is the call graph for this function:



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

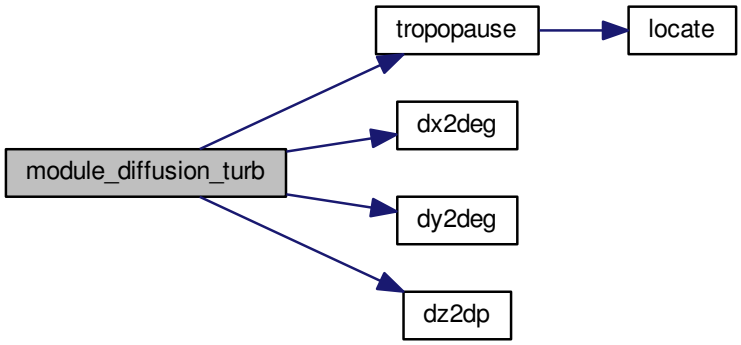
Calculate turbulent diffusion.

Definition at line 618 of file trac.c.

```
00623                 {
00624
00625   double dx, dz, pt, p0, p1, w;
00626
00627   /* Get tropopause pressure... */
00628   pt = tropopause(atm->time[ip], atm->lat[ip]);
00629
00630   /* Get weighting factor... */
00631   p1 = pt * 0.866877899;
00632   p0 = pt / 0.866877899;
00633   if (atm->p[ip] > p0)
00634     w = 1;
00635   else if (atm->p[ip] < p1)
00636     w = 0;
00637   else
00638     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00639
00640   /* Set diffusivitiy... */
00641   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00642   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00643
00644   /* Horizontal turbulent diffusion... */
00645   if (dx > 0) {
00646     atm->lon[ip]
00647       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00648                 / 1000., atm->lat[ip]);
00649     atm->lat[ip]
00650       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00651                 / 1000.);
00652   }
```

```
00653
00654   /* Vertical turbulent diffusion... */
00655   if (dz > 0)
00656     atm->p[ip]
00657       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00658                / 1000., atm->p[ip]);
00659 }
```

Here is the call graph for this function:



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

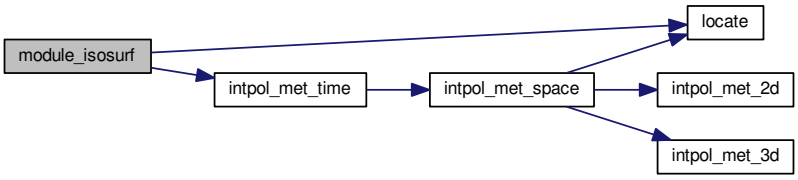
Force air parcels to stay on isosurface.

Definition at line 663 of file trac.c.

```
00668           {
00669
00670   static double *iso, *ps, t, *ts;
00671
00672   static int idx, ip2, n, nb = 100000;
00673
00674   FILE *in;
00675
00676   char line[LEN];
00677
00678   /* Check control parameter... */
00679   if (ctl->isosurf < 1 || ctl->isosurf > 4)
00680     return;
00681
00682   /* Initialize... */
00683   if (ip < 0) {
00684
00685     /* Allocate... */
00686     ALLOC(iso, double,
00687           NP);
00688     ALLOC(ps, double,
00689           nb);
00690     ALLOC(ts, double,
00691           nb);
00692
00693     /* Save pressure... */
00694     if (ctl->isosurf == 1)
00695       for (ip2 = 0; ip2 < atm->np; ip2++)
00696         iso[ip2] = atm->p[ip2];
00697
00698     /* Save density... */
00699     else if (ctl->isosurf == 2)
00700       for (ip2 = 0; ip2 < atm->np; ip2++) {
00701         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
```

```
00702                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00703                         NULL, NULL, NULL);
00704         iso[ip2] = atm->p[ip2] / t;
00705       }
00706
00707     /* Save potential temperature... */
00708     else if (ctl->isosurf == 3)
00709       for (ip2 = 0; ip2 < atm->np; ip2++) {
00710         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00711                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00712                         NULL, NULL, NULL);
00713         iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00714       }
00715
00716     /* Read balloon pressure data... */
00717     else if (ctl->isosurf == 4) {
00718
00719       /* Write info... */
00720       printf("Read balloon pressure data: %s\n", ctl->balloon);
00721
00722       /* Open file... */
00723       if (!(in = fopen(ctl->balloon, "r")))
00724         ERRMSG("Cannot open file!");
00725
00726       /* Read pressure time series... */
00727       while (fgets(line, LEN, in))
00728         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00729           if ((++n) > 100000)
00730             ERRMSG("Too many data points!");
00731
00732       /* Check number of points... */
00733       if (n < 1)
00734         ERRMSG("Could not read any data!");
00735
00736       /* Close file... */
00737       fclose(in);
00738     }
00739
00740     /* Leave initialization... */
00741     return;
00742   }
00743
00744   /* Restore pressure... */
00745   if (ctl->isosurf == 1)
00746     atm->p[ip] = iso[ip];
00747
00748   /* Restore density... */
00749   else if (ctl->isosurf == 2) {
00750     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00751                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00752     atm->p[ip] = iso[ip] * t;
00753   }
00754
00755   /* Restore potential temperature... */
00756   else if (ctl->isosurf == 3) {
00757     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00758                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00759     atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00760   }
00761
00762   /* Interpolate pressure... */
00763   else if (ctl->isosurf == 4) {
00764     if (atm->time[ip] <= ts[0])
00765       atm->p[ip] = ps[0];
00766     else if (atm->time[ip] >= ts[n - 1])
00767       atm->p[ip] = ps[n - 1];
00768     else {
00769       idx = locate(ts, n, atm->time[ip]);
00770       atm->p[ip] = LIN(ts[idx], ps[idx],
00771                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00772     }
00773   }
00774 }
```

Here is the call graph for this function:



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

Interpolate meteorological data for air parcel positions.

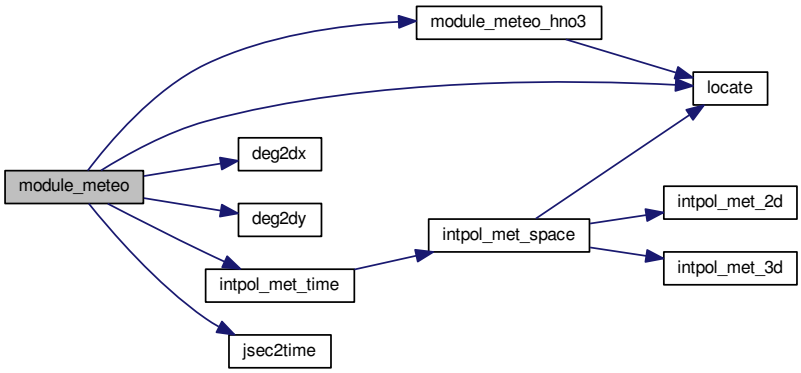
Definition at line 778 of file trac.c.

```
00783           {
00784
00785   static FILE *in;
00786
00787   static char filename[LEN], line[LEN];
00788
00789   static double lon[GX], lat[GY], var[GX][GY],
00790     rdum, rlat, rlat_old = -999, rlon, rvar;
00791
00792   static int year_old, mon_old, day_old, nlon, nlat;
00793
00794   double a, b, c, ps, p1, p_hno3, p_h2o, t, t1, u, u1, v, v1, w,
00795     x1, x2, h2o, o3, grad, vort, var0, var1;
00796
00797   int day, mon, year, idum, ilat, ilon;
00798
00799   /* Interpolate meteorological data... */
00800   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00801                   atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00802
00803   /* Set surface pressure... */
00804   if (ctl->qnt_ps >= 0)
00805     atm->q[ctl->qnt_ps][ip] = ps;
00806
00807   /* Set pressure... */
00808   if (ctl->qnt_p >= 0)
00809     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00810
00811   /* Set temperature... */
00812   if (ctl->qnt_t >= 0)
00813     atm->q[ctl->qnt_t][ip] = t;
00814
00815   /* Set zonal wind... */
00816   if (ctl->qnt_u >= 0)
00817     atm->q[ctl->qnt_u][ip] = u;
00818
00819   /* Set meridional wind... */
00820   if (ctl->qnt_v >= 0)
00821     atm->q[ctl->qnt_v][ip] = v;
00822
00823   /* Set vertical velocity... */
00824   if (ctl->qnt_w >= 0)
00825     atm->q[ctl->qnt_w][ip] = w;
00826
00827   /* Set water vapor vmr... */
00828   if (ctl->qnt_h2o >= 0)
00829     atm->q[ctl->qnt_h2o][ip] = h2o;
00830
00831   /* Set ozone vmr... */
00832   if (ctl->qnt_o3 >= 0)
00833     atm->q[ctl->qnt_o3][ip] = o3;
00834
00835   /* Calculate potential temperature... */
```

```
00836   if (ctl->qnt_theta >= 0)
00837     atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00838
00839   /* Calculate potential vorticity... */
00840   if (ctl->qnt_pv >= 0) {
00841
00842     /* Absolute vorticity... */
00843     vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00844     if (fabs(atm->lat[ip]) < 89.) {
00845       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00846                       (atm->lon[ip] >=
00847                        0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00848                       atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00849       vort += (v1 - v) / 1000.
00850         / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00851     }
00852     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00853                     (atm->lat[ip] >=
00854                      0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00855                     &u1, NULL, NULL, NULL, NULL);
00856     vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00857
00858     /* Potential temperature gradient... */
00859     p1 = 0.85 * atm->p[ip];
00860     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00861                     atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL);
00862     grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00863       / (100. * (p1 - atm->p[ip]));
00864
00865     /* Calculate PV... */
00866     atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00867   }
00868
00869   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00870   if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00871     atm->q[ctl->qnt_tice][ip] =
00872       -2663.5 /
00873       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00874        12.537);
00875
00876   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00877   if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
00878     if (ctl->psc_hno3 > 0)
00879       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00880     else
00881       p_hno3 = module_meteo_hno3(atm->time[ip], atm->lat[ip], atm->
      p[ip])
00882         * 1e-9 * atm->p[ip] / 1.333224;
00883     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00884     a = 0.009179 - 0.00088 * log10(p_h2o);
00885     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00886     c = -11397.0 / a;
00887     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00888     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00889     if (x1 > 0)
00890       atm->q[ctl->qnt_tnat][ip] = x1;
00891     if (x2 > 0)
00892       atm->q[ctl->qnt_tnat][ip] = x2;
00893   }
00894
00895   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00896   if (ctl->qnt_tsts >= 0) {
00897     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00898       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00899     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00900                                        + atm->q[ctl->qnt_tnat][ip]);
00901   }
00902
00903   /* Read variance data for current day... */
00904   if (ip == 0 && ctl->qnt_gw_var >= 0) {
00905     jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00906     if (year != year_old || mon != mon_old || day != day_old) {
00907       year_old = year;
00908       mon_old = mon;
00909       day_old = day;
00910       nlon = nlat = -1;
00911       sprintf(filename, "%s_%d_%02d_%02d.tab",
00912               ctl->gw_basename, year, mon, day);
00913       if ((in = fopen(filename, "r"))) {
00914         printf("Read gravity wave data: %s\n", filename);
00915         while (fgets(line, LEN, in)) {
00916           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00917             continue;
00918           if (rlat != rlat_old) {
00919             rlat_old = rlat;
00920             if ((++nlat) > GY)
```

```
00921               ERRMSG("Too many latitudes!");
00922             nlon = -1;
00923           }
00924           if ((++nlon) > GX)
00925             ERRMSG("Too many longitudes!");
00926           lon[nlon] = rlon;
00927           lat[nlat] = rlat;
00928           var[nlon][nlat] = GSL_MAX(0, rvar);
00929         }
00930         fclose(in);
00931         nlat++;
00932         nlon++;
00933       } else
00934         printf("Missing gravity wave data: %s\n", filename);
00935     }
00936   }
00937
00938   /* Interpolate variance data... */
00939   if (ctl->qnt_gw_var >= 0) {
00940     if (nlat >= 2 && nlon >= 2) {
00941       ilat = locate(lat, nlat, atm->lat[ip]);
00942       ilon = locate(lon, nlon, atm->lon[ip]);
00943       var0 = LIN(lat[ilat], var[ilon][ilat],
00944                  lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00945       var1 = LIN(lat[ilat], var[ilon + 1][ilat],
00946                  lat[ilat + 1], var[ilon + 1][ilat + 1], atm->lat[ip]);
00947       atm->q[ctl->qnt_gw_var][ip]
00948         = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00949     } else
00950       atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00951   }
00952 }
```

Here is the call graph for this function:



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

Auxiliary function for meteo module.

Definition at line 956 of file trac.c.

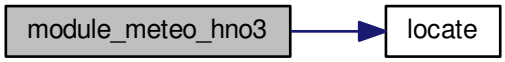
```
00959           {
00960
00961   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00962     9072000.00, 11664000.00, 14342400.00,
00963     16934400.00, 19612800.00, 22291200.00,
00964     24883200.00, 27561600.00, 30153600.00
00965   };
00966
00967   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00968     5, 15, 25, 35, 45, 55, 65, 75, 85
```

```
00969   };
00970
00971   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00972     31.6228, 46.4159, 68.1292, 100, 146.78
00973   };
00974
00975   static double hno3[12][18][10] = {
00976     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00977      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00978      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00979      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00980      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00981      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00982      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00983      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00984      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00985      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00986      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00987      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00988      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00989      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00990      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00991      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00992      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00993      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00994     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00995      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00996      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00997      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00998      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00999      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
01000      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
01001      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
01002      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
01003      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
01004      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
01005      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
01006      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
01007      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
01008      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
01009      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
01010      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
01011      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
01012     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
01013      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
01014      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
01015      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
01016      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
01017      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
01018      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
01019      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
01020      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
01021      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
01022      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
01023      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
01024      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
01025      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
01026      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
01027      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
01028      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
01029      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
01030     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
01031      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
01032      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
01033      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
01034      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
01035      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
01036      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
01037      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
01038      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
01039      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
01040      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
01041      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
01042      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
01043      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
01044      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
01045      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
01046      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
01047      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
01048     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
01049      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
01050      {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
01051      {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
01052      {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
01053      {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
01054      {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
01055      {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
```

```
01056       {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
01057       {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
01058       {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
01059       {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
01060       {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
01061       {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
01062       {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
01063       {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
01064       {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
01065       {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
01066      {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
01067       {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
01068       {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
01069       {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
01070       {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
01071       {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
01072       {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
01073       {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
01074       {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
01075       {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
01076       {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
01077       {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
01078       {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
01079       {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
01080       {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
01081       {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
01082       {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
01083       {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
01084      {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
01085       {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
01086       {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
01087       {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
01088       {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
01089       {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
01090       {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
01091       {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
01092       {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
01093       {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
01094       {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
01095       {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
01096       {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
01097       {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
01098       {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
01099       {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
01100       {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
01101       {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
01102      {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
01103       {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
01104       {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
01105       {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
01106       {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
01107       {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
01108       {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
01109       {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
01110       {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
01111       {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
01112       {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
01113       {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
01114       {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
01115       {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
01116       {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
01117       {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
01118       {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
01119       {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
01120      {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
01121       {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
01122       {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
01123       {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
01124       {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
01125       {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
01126       {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
01127       {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
01128       {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
01129       {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
01130       {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
01131       {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
01132       {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
01133       {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
01134       {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
01135       {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
01136       {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
01137       {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
01138      {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
01139       {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
01140       {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
01141       {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
01142       {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
```

```
01143       {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
01144       {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
01145       {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
01146       {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
01147       {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
01148       {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
01149       {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
01150       {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
01151       {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
01152       {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
01153       {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
01154       {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
01155       {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
01156     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
01157       {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
01158       {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
01159       {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
01160       {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
01161       {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
01162       {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
01163       {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
01164       {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
01165       {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
01166       {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
01167       {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
01168       {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
01169       {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
01170       {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
01171       {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
01172       {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
01173       {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
01174     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
01175       {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
01176       {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
01177       {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
01178       {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
01179       {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
01180       {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
01181       {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
01182       {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
01183       {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
01184       {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
01185       {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
01186       {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
01187       {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
01188       {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
01189       {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
01190       {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
01191       {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
01192   };
01193
01194   double aux00, aux01, aux10, aux11, sec;
01195
01196   int ilat, ip, isec;
01197
01198   /* Get seconds since begin of year... */
01199   sec = fmod(t, 365.25 * 86400.);
01200
01201   /* Get indices... */
01202   ilat = locate(lats, 18, lat);
01203   ip = locate(ps, 10, p);
01204   isec = locate(secs, 12, sec);
01205
01206   /* Interpolate... */
01207   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
01208               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
01209   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
01210               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
01211   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
01212               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
01213   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
01214               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
01215   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
01216   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
01217   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
01218 }
```

Here is the call graph for this function:



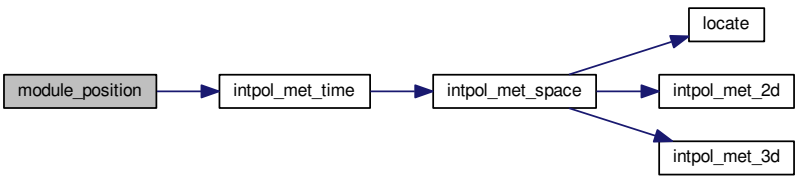
5.31.2.9 void module_position (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

Check position of air parcels.

Definition at line 1222 of file trac.c.

```
01226           {
01227
01228   double ps;
01229
01230   /* Calculate modulo... */
01231   atm->lon[ip] = fmod(atm->lon[ip], 360);
01232   atm->lat[ip] = fmod(atm->lat[ip], 360);
01233
01234   /* Check latitude... */
01235   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01236     if (atm->lat[ip] > 90) {
01237       atm->lat[ip] = 180 - atm->lat[ip];
01238       atm->lon[ip] += 180;
01239     }
01240     if (atm->lat[ip] < -90) {
01241       atm->lat[ip] = -180 - atm->lat[ip];
01242       atm->lon[ip] += 180;
01243     }
01244   }
01245
01246   /* Check longitude... */
01247   while (atm->lon[ip] < -180)
01248     atm->lon[ip] += 360;
01249   while (atm->lon[ip] >= 180)
01250     atm->lon[ip] -= 360;
01251
01252   /* Get surface pressure... */
01253   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01254                   atm->lon[ip], atm->lat[ip], &ps, NULL,
01255                   NULL, NULL, NULL, NULL, NULL);
01256
01257   /* Check pressure... */
01258   if (atm->p[ip] > ps)
01259     atm->p[ip] = ps;
01260   else if (atm->p[ip] < met0->p[met0->np - 1])
01261     atm->p[ip] = met0->p[met0->np - 1];
01262 }
```

Here is the call graph for this function:



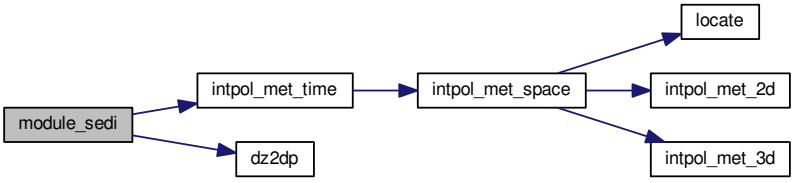
5.31.2.10 void module_sedi (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate sedimentation of air parcels.

Definition at line 1266 of file trac.c.

```
01272             {
01273
01274   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
01275   const double A = 1.249, B = 0.42, C = 0.87;
01276
01277   /* Specific gas constant for dry air [J/(kg K)]: */
01278   const double R = 287.058;
01279
01280   /* Average mass of an air molecule [kg/molec]: */
01281   const double m = 4.8096e-26;
01282
01283   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01284
01285   /* Check if parameters are available... */
01286   if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
01287     return;
01288
01289   /* Convert units... */
01290   p = 100 * atm->p[ip];
01291   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01292   rho_p = atm->q[ctl->qnt_rho][ip];
01293
01294   /* Get temperature... */
01295   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
01296                   atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
01297
01298   /* Density of dry air... */
01299   rho = p / (R * T);
01300
01301   /* Dynamic viscosity of air... */
01302   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01303
01304   /* Thermal velocity of an air molecule... */
01305   v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
01306
01307   /* Mean free path of an air molecule... */
01308   lambda = 2 * eta / (rho * v);
01309
01310   /* Knudsen number for air... */
01311   K = lambda / r_p;
01312
01313   /* Cunningham slip-flow correction... */
01314   G = 1 + K * (A + B * exp(-C / K));
01315
01316   /* Sedimentation (fall) velocity... */
01317   v_p =
01318     2. * gsl_pow_2(r_p) * (rho_p -
01319                            rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
01320
01321   /* Calculate pressure change... */
01322   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01323 }
```

Here is the call graph for this function:



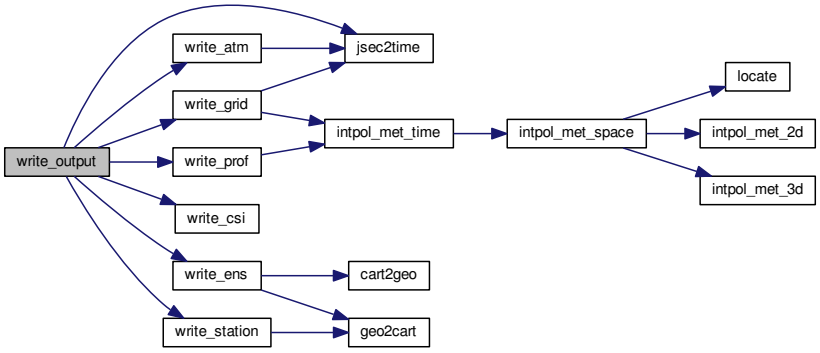
5.31.2.11 void write_output (const char * *dirname,* ctl_t * *ctl,* met_t * *met0,* met_t * *met1,* atm_t * *atm,* double *t*)

Write simulation output.

Definition at line 1327 of file trac.c.

```
01333             {
01334
01335   char filename[LEN];
01336
01337   double r;
01338
01339   int year, mon, day, hour, min, sec;
01340
01341   /* Get time... */
01342   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01343
01344   /* Write atmospheric data... */
01345   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01346     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01347             dirname, ctl->atm_basename, year, mon, day, hour, min);
01348     write_atm(filename, ctl, atm, t);
01349   }
01350
01351   /* Write CSI data... */
01352   if (ctl->csi_basename[0] != '-') {
01353     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01354     write_csi(filename, ctl, atm, t);
01355   }
01356
01357   /* Write ensemble data... */
01358   if (ctl->ens_basename[0] != '-') {
01359     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01360     write_ens(filename, ctl, atm, t);
01361   }
01362
01363   /* Write gridded data... */
01364   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01365     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01366             dirname, ctl->grid_basename, year, mon, day, hour, min);
01367     write_grid(filename, ctl, met0, met1, atm, t);
01368   }
01369
01370   /* Write profile data... */
01371   if (ctl->prof_basename[0] != '-') {
01372     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01373     write_prof(filename, ctl, met0, met1, atm, t);
01374   }
01375
01376   /* Write station data... */
01377   if (ctl->stat_basename[0] != '-') {
01378     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01379     write_station(filename, ctl, atm, t);
01380   }
01381 }
```

Here is the call graph for this function:



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

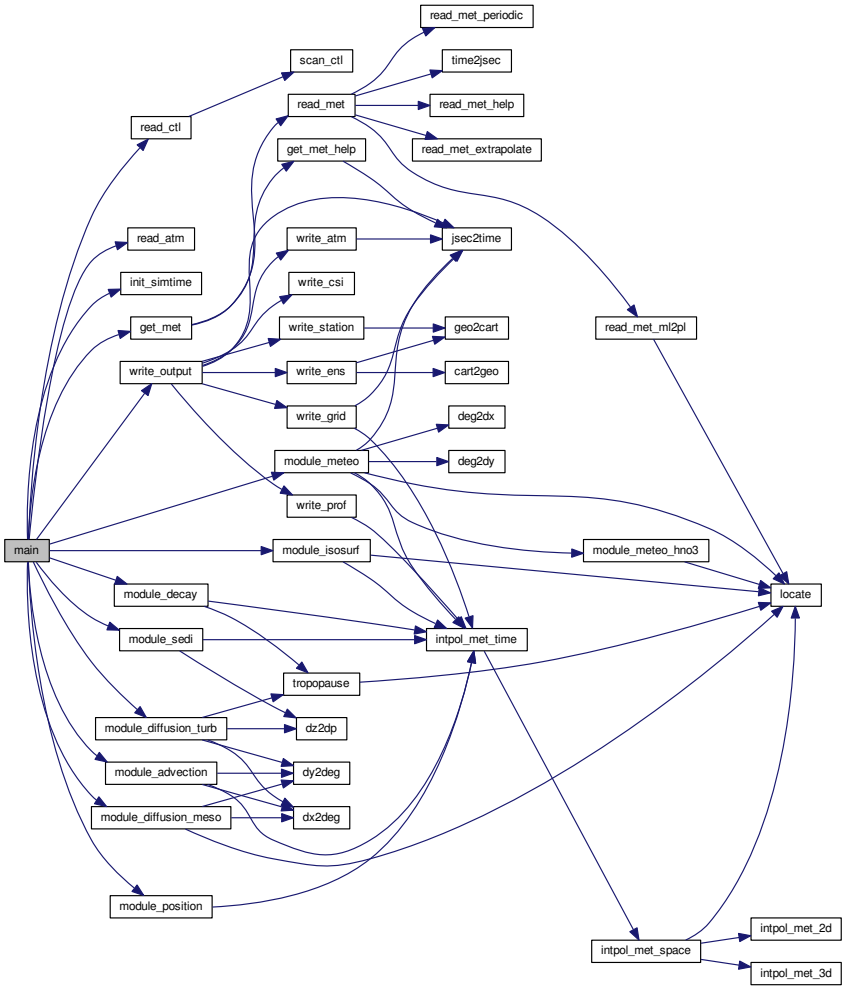
Definition at line 166 of file trac.c.

```
00168                {
00169
00170   ctl_t ctl;
00171
00172   atm_t *atm;
00173
00174   met_t *met0, *met1;
00175
00176   gsl_rng *rng[NTHREADS];
00177
00178   FILE *dirlist;
00179
00180   char dirname[LEN], filename[LEN];
00181
00182   double *dt, t, t0;
00183
00184   int i, ip, ntask = 0, rank = 0, size = 1;
00185
00186 #ifdef MPI
00187   /* Initialize MPI... */
00188   MPI_Init(&argc, &argv);
00189   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00190   MPI_Comm_size(MPI_COMM_WORLD, &size);
00191 #endif
00192
00193   /* Check arguments... */
00194   if (argc < 5)
00195     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00196
00197   /* Open directory list... */
00198   if (!(dirlist = fopen(argv[1], "r")))
00199     ERRMSG("Cannot open directory list!");
00200
00201   /* Loop over directories... */
00202   while (fscanf(dirlist, "%s", dirname) != EOF) {
00203
00204     /* MPI parallelization... */
00205     if ((++ntask) % size != rank)
00206       continue;
00207
00208     /* ------------------------------------------------------------
00209        Initialize model run...
00210        ------------------------------------------------------------ */
00211
00212     /* Set timers... */
00213     START_TIMER(TIMER_TOTAL);
00214     START_TIMER(TIMER_INIT);
00215
00216     /* Allocate... */
00217     ALLOC(atm, atm_t, 1);
00218     ALLOC(met0, met_t, 1);
00219     ALLOC(met1, met_t, 1);
00220     ALLOC(dt, double,
00221           NP);
00222
00223     /* Read control parameters... */
00224     sprintf(filename, "%s/%s", dirname, argv[2]);
00225     read_ctl(filename, argc, argv, &ctl);
00226
00227     /* Initialize random number generators... */
00228     gsl_rng_env_setup();
00229     for (i = 0; i < NTHREADS; i++)
00230       rng[i] = gsl_rng_alloc(gsl_rng_default);
00231
00232     /* Read atmospheric data... */
00233     sprintf(filename, "%s/%s", dirname, argv[3]);
00234     read_atm(filename, &ctl, atm);
00235
00236     /* Get simulation time interval... */
00237     init_simtime(&ctl, atm);
00238
00239     /* Get rounded start time... */
00240     if (ctl.direction == 1)
00241       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00242     else
00243       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00244
00245     /* Set timers... */
00246     STOP_TIMER(TIMER_INIT);
00247
```

```
00248     /* ----------------------------------------------------------
00249        Loop over timesteps...
00250        ---------------------------------------------------------- */
00251
00252     /* Loop over timesteps... */
00253     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00254          t += ctl.direction * ctl.dt_mod) {
00255
00256       /* Adjust length of final time step... */
00257       if (ctl.direction * (t - ctl.t_stop) > 0)
00258         t = ctl.t_stop;
00259
00260       /* Set time steps for air parcels... */
00261       for (ip = 0; ip < atm->np; ip++)
00262         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00263              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00264              && ctl.direction * (atm->time[ip] - t) < 0))
00265           dt[ip] = t - atm->time[ip];
00266         else
00267           dt[ip] = GSL_NAN;
00268
00269       /* Get meteorological data... */
00270       START_TIMER(TIMER_INPUT);
00271       get_met(&ctl, argv[4], t, met0, met1);
00272       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[1]) * 111132. / 150.
00273           || ctl.dt_mod > fabs(met1->lon[1] - met1->lon[1]) * 111132. / 150.)
00274         printf("Warning: Time step DT_MOD violates the CFL criterion!\n");
00275       STOP_TIMER(TIMER_INPUT);
00276
00277       /* Initialize isosurface... */
00278       START_TIMER(TIMER_ISOSURF);
00279       if (t == t0)
00280         module_isosurf(&ctl, met0, met1, atm, -1);
00281       STOP_TIMER(TIMER_ISOSURF);
00282
00283       /* Advection... */
00284       START_TIMER(TIMER_ADVECT);
00285 #pragma omp parallel for default(shared) private(ip)
00286       for (ip = 0; ip < atm->np; ip++)
00287         if (gsl_finite(dt[ip]))
00288           module_advection(met0, met1, atm, ip, dt[ip]);
00289       STOP_TIMER(TIMER_ADVECT);
00290
00291       /* Turbulent diffusion... */
00292       START_TIMER(TIMER_DIFFTURB);
00293 #pragma omp parallel for default(shared) private(ip)
00294       for (ip = 0; ip < atm->np; ip++)
00295         if (gsl_finite(dt[ip]))
00296           module_diffusion_turb(&ctl, atm, ip, dt[ip],
00297                                 rng[omp_get_thread_num()]);
00298       STOP_TIMER(TIMER_DIFFTURB);
00299
00300       /* Mesoscale diffusion... */
00301       START_TIMER(TIMER_DIFFMESO);
00302 #pragma omp parallel for default(shared) private(ip)
00303       for (ip = 0; ip < atm->np; ip++)
00304         if (gsl_finite(dt[ip]))
00305           module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00306                                 rng[omp_get_thread_num()]);
00307       STOP_TIMER(TIMER_DIFFMESO);
00308
00309       /* Sedimentation... */
00310       START_TIMER(TIMER_SEDI);
00311 #pragma omp parallel for default(shared) private(ip)
00312       for (ip = 0; ip < atm->np; ip++)
00313         if (gsl_finite(dt[ip]))
00314           module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00315       STOP_TIMER(TIMER_SEDI);
00316
00317       /* Isosurface... */
00318       START_TIMER(TIMER_ISOSURF);
00319 #pragma omp parallel for default(shared) private(ip)
00320       for (ip = 0; ip < atm->np; ip++)
00321         module_isosurf(&ctl, met0, met1, atm, ip);
00322       STOP_TIMER(TIMER_ISOSURF);
00323
00324       /* Position... */
00325       START_TIMER(TIMER_POSITION);
00326 #pragma omp parallel for default(shared) private(ip)
00327       for (ip = 0; ip < atm->np; ip++)
00328         module_position(met0, met1, atm, ip);
00329       STOP_TIMER(TIMER_POSITION);
00330
00331       /* Meteorological data... */
00332       START_TIMER(TIMER_METEO);
00333       module_meteo(&ctl, met0, met1, atm, 0);
00334 #pragma omp parallel for default(shared) private(ip)
```

```
00335       for (ip = 1; ip < atm->np; ip++)
00336         module_meteo(&ctl, met0, met1, atm, ip);
00337       STOP_TIMER(TIMER_METEO);
00338
00339       /* Decay... */
00340       START_TIMER(TIMER_DECAY);
00341 #pragma omp parallel for default(shared) private(ip)
00342       for (ip = 0; ip < atm->np; ip++)
00343         if (gsl_finite(dt[ip]))
00344           module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00345       STOP_TIMER(TIMER_DECAY);
00346
00347       /* Write output... */
00348       START_TIMER(TIMER_OUTPUT);
00349       write_output(dirname, &ctl, met0, met1, atm, t);
00350       STOP_TIMER(TIMER_OUTPUT);
00351     }
00352
00353     /* ------------------------------------------------------------
00354        Finalize model run...
00355        ------------------------------------------------------------ */
00356
00357     /* Report timers... */
00358     STOP_TIMER(TIMER_TOTAL);
00359     PRINT_TIMER(TIMER_TOTAL);
00360     PRINT_TIMER(TIMER_INIT);
00361     PRINT_TIMER(TIMER_INPUT);
00362     PRINT_TIMER(TIMER_OUTPUT);
00363     PRINT_TIMER(TIMER_ADVECT);
00364     PRINT_TIMER(TIMER_DECAY);
00365     PRINT_TIMER(TIMER_DIFFMESO);
00366     PRINT_TIMER(TIMER_DIFFTURB);
00367     PRINT_TIMER(TIMER_ISOSURF);
00368     PRINT_TIMER(TIMER_METEO);
00369     PRINT_TIMER(TIMER_POSITION);
00370     PRINT_TIMER(TIMER_SEDI);
00371
00372     /* Report memory usage... */
00373     printf("MEMORY_ATM = %g MByte\n", 2. * sizeof(atm_t) / 1024. / 1024.);
00374     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00375     printf("MEMORY_DYNAMIC = %g MByte\n",
00376            NP * sizeof(double) / 1024. / 1024.);
00377     printf("MEMORY_STATIC = %g MByte\n",
00378            (((EX + EY) + (2 + NQ) * GX * GY * GZ) * sizeof(double)
00379             + (EX * EY + EX * EY * EP) * sizeof(float)
00380             + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00381
00382     /* Report problem size... */
00383     printf("SIZE_NP = %d\n", atm->np);
00384     printf("SIZE_TASKS = %d\n", size);
00385     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00386
00387     /* Free random number generators... */
00388     for (i = 0; i < NTHREADS; i++)
00389       gsl_rng_free(rng[i]);
00390
00391     /* Free... */
00392     free(atm);
00393     free(met0);
00394     free(met1);
00395     free(dt);
00396   }
00397
00398 #ifdef MPI
00399   /* Finalize MPI... */
00400   MPI_Finalize();
00401 #endif
00402
00403   return EXIT_SUCCESS;
00404 }
```

Here is the call graph for this function:



5.32 trac.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
00030
```

```
00031 /* ------------------------------------------------------------
00032    Defines...
00033    ------------------------------------------------------------ */
00034
00036 #define TIMER_TOTAL 0
00037
00039 #define TIMER_INIT 1
00040
00042 #define TIMER_INPUT 2
00043
00045 #define TIMER_OUTPUT 3
00046
00048 #define TIMER_ADVECT 4
00049
00051 #define TIMER_DECAY 5
00052
00054 #define TIMER_DIFFMESO 6
00055
00057 #define TIMER_DIFFTURB 7
00058
00060 #define TIMER_ISOSURF 8
00061
00063 #define TIMER_METEO 9
00064
00066 #define TIMER_POSITION 10
00067
00069 #define TIMER_SEDI 11
00070
00071 /* ------------------------------------------------------------
00072    Functions...
00073    ------------------------------------------------------------ */
00074
00076 void init_simtime(
00077   ctl_t * ctl,
00078   atm_t * atm);
00079
00081 void module_advection(
00082   met_t * met0,
00083   met_t * met1,
00084   atm_t * atm,
00085   int ip,
00086   double dt);
00087
00089 void module_decay(
00090   ctl_t * ctl,
00091   met_t * met0,
00092   met_t * met1,
00093   atm_t * atm,
00094   int ip,
00095   double dt);
00096
00098 void module_diffusion_meso(
00099   ctl_t * ctl,
00100   met_t * met0,
00101   met_t * met1,
00102   atm_t * atm,
00103   int ip,
00104   double dt,
00105   gsl_rng * rng);
00106
00108 void module_diffusion_turb(
00109   ctl_t * ctl,
00110   atm_t * atm,
00111   int ip,
00112   double dt,
00113   gsl_rng * rng);
00114
00116 void module_isosurf(
00117   ctl_t * ctl,
00118   met_t * met0,
00119   met_t * met1,
00120   atm_t * atm,
00121   int ip);
00122
00124 void module_meteo(
00125   ctl_t * ctl,
00126   met_t * met0,
00127   met_t * met1,
00128   atm_t * atm,
00129   int ip);
00130
00132 double module_meteo_hno3(
00133   double t,
00134   double lat,
00135   double p);
00136
00138 void module_position(
```

```
00139   met_t * met0,
00140   met_t * met1,
00141   atm_t * atm,
00142   int ip);
00143
00145 void module_sedi(
00146   ctl_t * ctl,
00147   met_t * met0,
00148   met_t * met1,
00149   atm_t * atm,
00150   int ip,
00151   double dt);
00152
00154 void write_output(
00155   const char *dirname,
00156   ctl_t * ctl,
00157   met_t * met0,
00158   met_t * met1,
00159   atm_t * atm,
00160   double t);
00161
00162 /* ------------------------------------------------------------
00163    Main...
00164    ------------------------------------------------------------ */
00165
00166 int main(
00167   int argc,
00168   char *argv[]) {
00169
00170   ctl_t ctl;
00171
00172   atm_t *atm;
00173
00174   met_t *met0, *met1;
00175
00176   gsl_rng *rng[NTHREADS];
00177
00178   FILE *dirlist;
00179
00180   char dirname[LEN], filename[LEN];
00181
00182   double *dt, t, t0;
00183
00184   int i, ip, ntask = 0, rank = 0, size = 1;
00185
00186 #ifdef MPI
00187   /* Initialize MPI... */
00188   MPI_Init(&argc, &argv);
00189   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00190   MPI_Comm_size(MPI_COMM_WORLD, &size);
00191 #endif
00192
00193   /* Check arguments... */
00194   if (argc < 5)
00195     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00196
00197   /* Open directory list... */
00198   if (!(dirlist = fopen(argv[1], "r")))
00199     ERRMSG("Cannot open directory list!");
00200
00201   /* Loop over directories... */
00202   while (fscanf(dirlist, "%s", dirname) != EOF) {
00203
00204     /* MPI parallelization... */
00205     if ((++ntask) % size != rank)
00206       continue;
00207
00208     /* ------------------------------------------------------------
00209        Initialize model run...
00210        ------------------------------------------------------------ */
00211
00212     /* Set timers... */
00213     START_TIMER(TIMER_TOTAL);
00214     START_TIMER(TIMER_INIT);
00215
00216     /* Allocate... */
00217     ALLOC(atm, atm_t, 1);
00218     ALLOC(met0, met_t, 1);
00219     ALLOC(met1, met_t, 1);
00220     ALLOC(dt, double,
00221           NP);
00222
00223     /* Read control parameters... */
00224     sprintf(filename, "%s/%s", dirname, argv[2]);
00225     read_ctl(filename, argc, argv, &ctl);
00226
00227     /* Initialize random number generators... */
```

```
00228     gsl_rng_env_setup();
00229     for (i = 0; i < NTHREADS; i++)
00230       rng[i] = gsl_rng_alloc(gsl_rng_default);
00231
00232     /* Read atmospheric data... */
00233     sprintf(filename, "%s/%s", dirname, argv[3]);
00234     read_atm(filename, &ctl, atm);
00235
00236     /* Get simulation time interval... */
00237     init_simtime(&ctl, atm);
00238
00239     /* Get rounded start time... */
00240     if (ctl.direction == 1)
00241       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00242     else
00243       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00244
00245     /* Set timers... */
00246     STOP_TIMER(TIMER_INIT);
00247
00248     /* ------------------------------------------------------------
00249        Loop over timesteps...
00250        ------------------------------------------------------------ */
00251
00252     /* Loop over timesteps... */
00253     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00254          t += ctl.direction * ctl.dt_mod) {
00255
00256       /* Adjust length of final time step... */
00257       if (ctl.direction * (t - ctl.t_stop) > 0)
00258         t = ctl.t_stop;
00259
00260       /* Set time steps for air parcels... */
00261       for (ip = 0; ip < atm->np; ip++)
00262         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00263              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00264              && ctl.direction * (atm->time[ip] - t) < 0))
00265           dt[ip] = t - atm->time[ip];
00266         else
00267           dt[ip] = GSL_NAN;
00268
00269       /* Get meteorological data... */
00270       START_TIMER(TIMER_INPUT);
00271       get_met(&ctl, argv[4], t, met0, met1);
00272       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[1]) * 111132. / 150.
00273           || ctl.dt_mod > fabs(met1->lon[1] - met1->lon[1]) * 111132. / 150.)
00274         printf("Warning: Time step DT_MOD violates the CFL criterion!\n");
00275       STOP_TIMER(TIMER_INPUT);
00276
00277       /* Initialize isosurface... */
00278       START_TIMER(TIMER_ISOSURF);
00279       if (t == t0)
00280         module_isosurf(&ctl, met0, met1, atm, -1);
00281       STOP_TIMER(TIMER_ISOSURF);
00282
00283       /* Advection... */
00284       START_TIMER(TIMER_ADVECT);
00285 #pragma omp parallel for default(shared) private(ip)
00286       for (ip = 0; ip < atm->np; ip++)
00287         if (gsl_finite(dt[ip]))
00288           module_advection(met0, met1, atm, ip, dt[ip]);
00289       STOP_TIMER(TIMER_ADVECT);
00290
00291       /* Turbulent diffusion... */
00292       START_TIMER(TIMER_DIFFTURB);
00293 #pragma omp parallel for default(shared) private(ip)
00294       for (ip = 0; ip < atm->np; ip++)
00295         if (gsl_finite(dt[ip]))
00296           module_diffusion_turb(&ctl, atm, ip, dt[ip],
00297                                 rng[omp_get_thread_num()]);
00298       STOP_TIMER(TIMER_DIFFTURB);
00299
00300       /* Mesoscale diffusion... */
00301       START_TIMER(TIMER_DIFFMESO);
00302 #pragma omp parallel for default(shared) private(ip)
00303       for (ip = 0; ip < atm->np; ip++)
00304         if (gsl_finite(dt[ip]))
00305           module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00306                                 rng[omp_get_thread_num()]);
00307       STOP_TIMER(TIMER_DIFFMESO);
00308
00309       /* Sedimentation... */
00310       START_TIMER(TIMER_SEDI);
00311 #pragma omp parallel for default(shared) private(ip)
00312       for (ip = 0; ip < atm->np; ip++)
00313         if (gsl_finite(dt[ip]))
00314           module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
```

```
00315       STOP_TIMER(TIMER_SEDI);
00316
00317       /* Isosurface... */
00318       START_TIMER(TIMER_ISOSURF);
00319 #pragma omp parallel for default(shared) private(ip)
00320       for (ip = 0; ip < atm->np; ip++)
00321         module_isosurf(&ctl, met0, met1, atm, ip);
00322       STOP_TIMER(TIMER_ISOSURF);
00323
00324       /* Position... */
00325       START_TIMER(TIMER_POSITION);
00326 #pragma omp parallel for default(shared) private(ip)
00327       for (ip = 0; ip < atm->np; ip++)
00328         module_position(met0, met1, atm, ip);
00329       STOP_TIMER(TIMER_POSITION);
00330
00331       /* Meteorological data... */
00332       START_TIMER(TIMER_METEO);
00333       module_meteo(&ctl, met0, met1, atm, 0);
00334 #pragma omp parallel for default(shared) private(ip)
00335       for (ip = 1; ip < atm->np; ip++)
00336         module_meteo(&ctl, met0, met1, atm, ip);
00337       STOP_TIMER(TIMER_METEO);
00338
00339       /* Decay... */
00340       START_TIMER(TIMER_DECAY);
00341 #pragma omp parallel for default(shared) private(ip)
00342       for (ip = 0; ip < atm->np; ip++)
00343         if (gsl_finite(dt[ip]))
00344           module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00345       STOP_TIMER(TIMER_DECAY);
00346
00347       /* Write output... */
00348       START_TIMER(TIMER_OUTPUT);
00349       write_output(dirname, &ctl, met0, met1, atm, t);
00350       STOP_TIMER(TIMER_OUTPUT);
00351     }
00352
00353     /* ------------------------------------------------------------
00354        Finalize model run...
00355        ------------------------------------------------------------ */
00356
00357     /* Report timers... */
00358     STOP_TIMER(TIMER_TOTAL);
00359     PRINT_TIMER(TIMER_TOTAL);
00360     PRINT_TIMER(TIMER_INIT);
00361     PRINT_TIMER(TIMER_INPUT);
00362     PRINT_TIMER(TIMER_OUTPUT);
00363     PRINT_TIMER(TIMER_ADVECT);
00364     PRINT_TIMER(TIMER_DECAY);
00365     PRINT_TIMER(TIMER_DIFFMESO);
00366     PRINT_TIMER(TIMER_DIFFTURB);
00367     PRINT_TIMER(TIMER_ISOSURF);
00368     PRINT_TIMER(TIMER_METEO);
00369     PRINT_TIMER(TIMER_POSITION);
00370     PRINT_TIMER(TIMER_SEDI);
00371
00372     /* Report memory usage... */
00373     printf("MEMORY_ATM = %g MByte\n", 2. * sizeof(atm_t) / 1024. / 1024.);
00374     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00375     printf("MEMORY_DYNAMIC = %g MByte\n",
00376            NP * sizeof(double) / 1024. / 1024.);
00377     printf("MEMORY_STATIC = %g MByte\n",
00378            (((EX + EY) + (2 + NQ) * GX * GY * GZ) * sizeof(double)
00379             + (EX * EY + EX * EY * EP) * sizeof(float)
00380             + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00381
00382     /* Report problem size... */
00383     printf("SIZE_NP = %d\n", atm->np);
00384     printf("SIZE_TASKS = %d\n", size);
00385     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00386
00387     /* Free random number generators... */
00388     for (i = 0; i < NTHREADS; i++)
00389       gsl_rng_free(rng[i]);
00390
00391     /* Free... */
00392     free(atm);
00393     free(met0);
00394     free(met1);
00395     free(dt);
00396   }
00397
00398 #ifdef MPI
00399   /* Finalize MPI... */
00400   MPI_Finalize();
00401 #endif
```

```
00402
00403   return EXIT_SUCCESS;
00404 }
00405
00406 /*****************************************************************************/
00407
00408 void init_simtime(
00409   ctl_t * ctl,
00410   atm_t * atm) {
00411
00412   /* Set inital and final time... */
00413   if (ctl->direction == 1) {
00414     if (ctl->t_start < -1e99)
00415       ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00416     if (ctl->t_stop < -1e99)
00417       ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00418   } else if (ctl->direction == -1) {
00419     if (ctl->t_stop < -1e99)
00420       ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00421     if (ctl->t_start < -1e99)
00422       ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00423   }
00424
00425   /* Check time... */
00426   if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
00427     ERRMSG("Nothing to do!");
00428 }
00429
00430 /*****************************************************************************/
00431
00432 void module_advection(
00433   met_t * met0,
00434   met_t * met1,
00435   atm_t * atm,
00436   int ip,
00437   double dt) {
00438
00439   double v[3], xm[3];
00440
00441   /* Interpolate meteorological data... */
00442   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00443                   atm->lon[ip], atm->lat[ip], NULL, NULL,
00444                   &v[0], &v[1], &v[2], NULL, NULL);
00445
00446   /* Get position of the mid point... */
00447   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00448   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00449   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00450
00451   /* Interpolate meteorological data for mid point... */
00452   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00453                   xm[2], xm[0], xm[1], NULL, NULL,
00454                   &v[0], &v[1], &v[2], NULL, NULL);
00455
00456   /* Save new position... */
00457   atm->time[ip] += dt;
00458   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00459   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00460   atm->p[ip] += dt * v[2];
00461 }
00462
00463 /*****************************************************************************/
00464
00465 void module_decay(
00466   ctl_t * ctl,
00467   met_t * met0,
00468   met_t * met1,
00469   atm_t * atm,
00470   int ip,
00471   double dt) {
00472
00473   double ps, pt, tdec;
00474
00475   /* Check lifetime values... */
00476   if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00477     return;
00478
00479   /* Set constant lifetime... */
00480   if (ctl->tdec_trop == ctl->tdec_strat)
00481     tdec = ctl->tdec_trop;
00482
00483   /* Set altitude-dependent lifetime... */
00484   else {
00485
00486     /* Get surface pressure... */
00487     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
```

```
00488                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00489                     NULL, NULL, NULL, NULL, NULL);
00490
00491     /* Get tropopause pressure... */
00492     pt = tropopause(atm->time[ip], atm->lat[ip]);
00493
00494     /* Set lifetime... */
00495     if (atm->p[ip] <= pt)
00496       tdec = ctl->tdec_strat;
00497     else
00498       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00499   }
00500
00501   /* Calculate exponential decay... */
00502   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00503 }
00504
00505 /*****************************************************************************/
00506
00507 void module_diffusion_meso(
00508   ctl_t * ctl,
00509   met_t * met0,
00510   met_t * met1,
00511   atm_t * atm,
00512   int ip,
00513   double dt,
00514   gsl_rng * rng) {
00515
00516   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00517
00518   int ix, iy, iz;
00519
00520   /* Calculate mesoscale velocity fluctuations... */
00521   if (ctl->turb_meso > 0) {
00522
00523     /* Get indices... */
00524     ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00525     iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00526     iz = locate(met0->p, met0->np, atm->p[ip]);
00527
00528     /* Collect local wind data... */
00529     u[0] = met0->u[ix][iy][iz];
00530     u[1] = met0->u[ix + 1][iy][iz];
00531     u[2] = met0->u[ix][iy + 1][iz];
00532     u[3] = met0->u[ix + 1][iy + 1][iz];
00533     u[4] = met0->u[ix][iy][iz + 1];
00534     u[5] = met0->u[ix + 1][iy][iz + 1];
00535     u[6] = met0->u[ix][iy + 1][iz + 1];
00536     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00537
00538     v[0] = met0->v[ix][iy][iz];
00539     v[1] = met0->v[ix + 1][iy][iz];
00540     v[2] = met0->v[ix][iy + 1][iz];
00541     v[3] = met0->v[ix + 1][iy + 1][iz];
00542     v[4] = met0->v[ix][iy][iz + 1];
00543     v[5] = met0->v[ix + 1][iy][iz + 1];
00544     v[6] = met0->v[ix][iy + 1][iz + 1];
00545     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00546
00547     w[0] = met0->w[ix][iy][iz];
00548     w[1] = met0->w[ix + 1][iy][iz];
00549     w[2] = met0->w[ix][iy + 1][iz];
00550     w[3] = met0->w[ix + 1][iy + 1][iz];
00551     w[4] = met0->w[ix][iy][iz + 1];
00552     w[5] = met0->w[ix + 1][iy][iz + 1];
00553     w[6] = met0->w[ix][iy + 1][iz + 1];
00554     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00555
00556     /* Get indices... */
00557     ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00558     iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00559     iz = locate(met1->p, met1->np, atm->p[ip]);
00560
00561     /* Collect local wind data... */
00562     u[8] = met1->u[ix][iy][iz];
00563     u[9] = met1->u[ix + 1][iy][iz];
00564     u[10] = met1->u[ix][iy + 1][iz];
00565     u[11] = met1->u[ix + 1][iy + 1][iz];
00566     u[12] = met1->u[ix][iy][iz + 1];
00567     u[13] = met1->u[ix + 1][iy][iz + 1];
00568     u[14] = met1->u[ix][iy + 1][iz + 1];
00569     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00570
00571     v[8] = met1->v[ix][iy][iz];
00572     v[9] = met1->v[ix + 1][iy][iz];
00573     v[10] = met1->v[ix][iy + 1][iz];
```

```
00574     v[11] = met1->v[ix + 1][iy + 1][iz];
00575     v[12] = met1->v[ix][iy][iz + 1];
00576     v[13] = met1->v[ix + 1][iy][iz + 1];
00577     v[14] = met1->v[ix][iy + 1][iz + 1];
00578     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00579
00580     w[8] = met1->w[ix][iy][iz];
00581     w[9] = met1->w[ix + 1][iy][iz];
00582     w[10] = met1->w[ix][iy + 1][iz];
00583     w[11] = met1->w[ix + 1][iy + 1][iz];
00584     w[12] = met1->w[ix][iy][iz + 1];
00585     w[13] = met1->w[ix + 1][iy][iz + 1];
00586     w[14] = met1->w[ix][iy + 1][iz + 1];
00587     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00588
00589     /* Get standard deviations of local wind data... */
00590     usig = gsl_stats_sd(u, 1, 16);
00591     vsig = gsl_stats_sd(v, 1, 16);
00592     wsig = gsl_stats_sd(w, 1, 16);
00593
00594     /* Set temporal correlations for mesoscale fluctuations... */
00595     r = 1 - 2 * fabs(dt) / ctl->dt_met;
00596     rs = sqrt(1 - r * r);
00597
00598     /* Calculate mesoscale wind fluctuations... */
00599     atm->up[ip] =
00600       r * atm->up[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00601                                                        ctl->turb_meso * usig);
00602     atm->vp[ip] =
00603       r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00604                                                        ctl->turb_meso * vsig);
00605     atm->wp[ip] =
00606       r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00607                                                        ctl->turb_meso * wsig);
00608
00609     /* Calculate air parcel displacement... */
00610     atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00611     atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00612     atm->p[ip] += atm->wp[ip] * dt;
00613   }
00614 }
00615
00616 /*****************************************************************************/
00617
00618 void module_diffusion_turb(
00619   ctl_t * ctl,
00620   atm_t * atm,
00621   int ip,
00622   double dt,
00623   gsl_rng * rng) {
00624
00625   double dx, dz, pt, p0, p1, w;
00626
00627   /* Get tropopause pressure... */
00628   pt = tropopause(atm->time[ip], atm->lat[ip]);
00629
00630   /* Get weighting factor... */
00631   p1 = pt * 0.866877899;
00632   p0 = pt / 0.866877899;
00633   if (atm->p[ip] > p0)
00634     w = 1;
00635   else if (atm->p[ip] < p1)
00636     w = 0;
00637   else
00638     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00639
00640   /* Set diffusivitiy... */
00641   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00642   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00643
00644   /* Horizontal turbulent diffusion... */
00645   if (dx > 0) {
00646     atm->lon[ip]
00647       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00648                 / 1000., atm->lat[ip]);
00649     atm->lat[ip]
00650       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00651                 / 1000.);
00652   }
00653
00654   /* Vertical turbulent diffusion... */
00655   if (dz > 0)
00656     atm->p[ip]
00657       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00658                / 1000., atm->p[ip]);
00659 }
00660
```

```
00661 /*****************************************************************************/
00662
00663 void module_isosurf(
00664   ctl_t * ctl,
00665   met_t * met0,
00666   met_t * met1,
00667   atm_t * atm,
00668   int ip) {
00669
00670   static double *iso, *ps, t, *ts;
00671
00672   static int idx, ip2, n, nb = 100000;
00673
00674   FILE *in;
00675
00676   char line[LEN];
00677
00678   /* Check control parameter... */
00679   if (ctl->isosurf < 1 || ctl->isosurf > 4)
00680     return;
00681
00682   /* Initialize... */
00683   if (ip < 0) {
00684
00685     /* Allocate... */
00686     ALLOC(iso, double,
00687           NP);
00688     ALLOC(ps, double,
00689           nb);
00690     ALLOC(ts, double,
00691           nb);
00692
00693     /* Save pressure... */
00694     if (ctl->isosurf == 1)
00695       for (ip2 = 0; ip2 < atm->np; ip2++)
00696         iso[ip2] = atm->p[ip2];
00697
00698     /* Save density... */
00699     else if (ctl->isosurf == 2)
00700       for (ip2 = 0; ip2 < atm->np; ip2++) {
00701         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00702                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00703                         NULL, NULL, NULL);
00704         iso[ip2] = atm->p[ip2] / t;
00705       }
00706
00707     /* Save potential temperature... */
00708     else if (ctl->isosurf == 3)
00709       for (ip2 = 0; ip2 < atm->np; ip2++) {
00710         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00711                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00712                         NULL, NULL, NULL);
00713         iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00714       }
00715
00716     /* Read balloon pressure data... */
00717     else if (ctl->isosurf == 4) {
00718
00719       /* Write info... */
00720       printf("Read balloon pressure data: %s\n", ctl->balloon);
00721
00722       /* Open file... */
00723       if (!(in = fopen(ctl->balloon, "r")))
00724         ERRMSG("Cannot open file!");
00725
00726       /* Read pressure time series... */
00727       while (fgets(line, LEN, in))
00728         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00729           if ((++n) > 100000)
00730             ERRMSG("Too many data points!");
00731
00732       /* Check number of points... */
00733       if (n < 1)
00734         ERRMSG("Could not read any data!");
00735
00736       /* Close file... */
00737       fclose(in);
00738     }
00739
00740     /* Leave initialization... */
00741     return;
00742   }
00743
00744   /* Restore pressure... */
00745   if (ctl->isosurf == 1)
00746     atm->p[ip] = iso[ip];
00747
```

```
00748   /* Restore density... */
00749   else if (ctl->isosurf == 2) {
00750     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00751                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00752     atm->p[ip] = iso[ip] * t;
00753   }
00754
00755   /* Restore potential temperature... */
00756   else if (ctl->isosurf == 3) {
00757     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00758                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00759     atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00760   }
00761
00762   /* Interpolate pressure... */
00763   else if (ctl->isosurf == 4) {
00764     if (atm->time[ip] <= ts[0])
00765       atm->p[ip] = ps[0];
00766     else if (atm->time[ip] >= ts[n - 1])
00767       atm->p[ip] = ps[n - 1];
00768     else {
00769       idx = locate(ts, n, atm->time[ip]);
00770       atm->p[ip] = LIN(ts[idx], ps[idx],
00771                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00772     }
00773   }
00774 }
00775
00776 /*****************************************************************************/
00777
00778 void module_meteo(
00779   ctl_t * ctl,
00780   met_t * met0,
00781   met_t * met1,
00782   atm_t * atm,
00783   int ip) {
00784
00785   static FILE *in;
00786
00787   static char filename[LEN], line[LEN];
00788
00789   static double lon[GX], lat[GY], var[GX][GY],
00790     rdum, rlat, rlat_old = -999, rlon, rvar;
00791
00792   static int year_old, mon_old, day_old, nlon, nlat;
00793
00794   double a, b, c, ps, p1, p_hno3, p_h2o, t, t1, u, u1, v, v1, w,
00795     x1, x2, h2o, o3, grad, vort, var0, var1;
00796
00797   int day, mon, year, idum, ilat, ilon;
00798
00799   /* Interpolate meteorological data... */
00800   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00801                   atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00802
00803   /* Set surface pressure... */
00804   if (ctl->qnt_ps >= 0)
00805     atm->q[ctl->qnt_ps][ip] = ps;
00806
00807   /* Set pressure... */
00808   if (ctl->qnt_p >= 0)
00809     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00810
00811   /* Set temperature... */
00812   if (ctl->qnt_t >= 0)
00813     atm->q[ctl->qnt_t][ip] = t;
00814
00815   /* Set zonal wind... */
00816   if (ctl->qnt_u >= 0)
00817     atm->q[ctl->qnt_u][ip] = u;
00818
00819   /* Set meridional wind... */
00820   if (ctl->qnt_v >= 0)
00821     atm->q[ctl->qnt_v][ip] = v;
00822
00823   /* Set vertical velocity... */
00824   if (ctl->qnt_w >= 0)
00825     atm->q[ctl->qnt_w][ip] = w;
00826
00827   /* Set water vapor vmr... */
00828   if (ctl->qnt_h2o >= 0)
00829     atm->q[ctl->qnt_h2o][ip] = h2o;
00830
00831   /* Set ozone vmr... */
```

```
00832   if (ctl->qnt_o3 >= 0)
00833     atm->q[ctl->qnt_o3][ip] = o3;
00834
00835   /* Calculate potential temperature... */
00836   if (ctl->qnt_theta >= 0)
00837     atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00838
00839   /* Calculate potential vorticity... */
00840   if (ctl->qnt_pv >= 0) {
00841
00842     /* Absolute vorticity... */
00843     vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00844     if (fabs(atm->lat[ip]) < 89.) {
00845       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00846                       (atm->lon[ip] >=
00847                        0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00848                       atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00849       vort += (v1 - v) / 1000.
00850         / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00851     }
00852     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00853                     (atm->lat[ip] >=
00854                      0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00855                     &u1, NULL, NULL, NULL, NULL);
00856     vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00857
00858     /* Potential temperature gradient... */
00859     p1 = 0.85 * atm->p[ip];
00860     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00861                     atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL);
00862     grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00863       / (100. * (p1 - atm->p[ip]));
00864
00865     /* Calculate PV... */
00866     atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00867   }
00868
00869   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00870   if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00871     atm->q[ctl->qnt_tice][ip] =
00872       -2663.5 /
00873       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00874        12.537);
00875
00876   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00877   if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
00878     if (ctl->psc_hno3 > 0)
00879       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00880     else
00881       p_hno3 = module_meteo_hno3(atm->time[ip], atm->lat[ip], atm->
      p[ip])
00882         * 1e-9 * atm->p[ip] / 1.333224;
00883     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00884     a = 0.009179 - 0.00088 * log10(p_h2o);
00885     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00886     c = -11397.0 / a;
00887     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00888     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00889     if (x1 > 0)
00890       atm->q[ctl->qnt_tnat][ip] = x1;
00891     if (x2 > 0)
00892       atm->q[ctl->qnt_tnat][ip] = x2;
00893   }
00894
00895   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00896   if (ctl->qnt_tsts >= 0) {
00897     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00898       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00899     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00900                                        + atm->q[ctl->qnt_tnat][ip]);
00901   }
00902
00903   /* Read variance data for current day... */
00904   if (ip == 0 && ctl->qnt_gw_var >= 0) {
00905     jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00906     if (year != year_old || mon != mon_old || day != day_old) {
00907       year_old = year;
00908       mon_old = mon;
00909       day_old = day;
00910       nlon = nlat = -1;
00911       sprintf(filename, "%s_%d_%02d_%02d.tab",
00912               ctl->gw_basename, year, mon, day);
00913       if ((in = fopen(filename, "r"))) {
00914         printf("Read gravity wave data: %s\n", filename);
00915         while (fgets(line, LEN, in)) {
00916           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
```

```
00917             continue;
00918           if (rlat != rlat_old) {
00919             rlat_old = rlat;
00920             if ((++nlat) > GY)
00921               ERRMSG("Too many latitudes!");
00922             nlon = -1;
00923           }
00924           if ((++nlon) > GX)
00925             ERRMSG("Too many longitudes!");
00926           lon[nlon] = rlon;
00927           lat[nlat] = rlat;
00928           var[nlon][nlat] = GSL_MAX(0, rvar);
00929         }
00930         fclose(in);
00931         nlat++;
00932         nlon++;
00933       } else
00934         printf("Missing gravity wave data: %s\n", filename);
00935     }
00936   }
00937
00938   /* Interpolate variance data... */
00939   if (ctl->qnt_gw_var >= 0) {
00940     if (nlat >= 2 && nlon >= 2) {
00941       ilat = locate(lat, nlat, atm->lat[ip]);
00942       ilon = locate(lon, nlon, atm->lon[ip]);
00943       var0 = LIN(lat[ilat], var[ilon][ilat],
00944                  lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00945       var1 = LIN(lat[ilat], var[ilon + 1][ilat],
00946                  lat[ilat + 1], var[ilon + 1][ilat + 1], atm->lat[ip]);
00947       atm->q[ctl->qnt_gw_var][ip]
00948         = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00949     } else
00950       atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00951   }
00952 }
00953
00954 /*****************************************************************************/
00955
00956 double module_meteo_hno3(
00957   double t,
00958   double lat,
00959   double p) {
00960
00961   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00962     9072000.00, 11664000.00, 14342400.00,
00963     16934400.00, 19612800.00, 22291200.00,
00964     24883200.00, 27561600.00, 30153600.00
00965   };
00966
00967   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00968     5, 15, 25, 35, 45, 55, 65, 75, 85
00969   };
00970
00971   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00972     31.6228, 46.4159, 68.1292, 100, 146.78
00973   };
00974
00975   static double hno3[12][18][10] = {
00976     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00977      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00978      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00979      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00980      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00981      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00982      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00983      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00984      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00985      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00986      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00987      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00988      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00989      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00990      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00991      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00992      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00993      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00994     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00995      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00996      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00997      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00998      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00999      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
01000      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
01001      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
01002      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
01003      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
```

```
01004      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
01005      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
01006      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
01007      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
01008      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
01009      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
01010      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
01011      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
01012     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
01013      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
01014      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
01015      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
01016      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
01017      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
01018      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
01019      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
01020      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
01021      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
01022      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
01023      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
01024      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
01025      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
01026      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
01027      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
01028      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
01029      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
01030     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
01031      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
01032      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
01033      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
01034      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
01035      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
01036      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
01037      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
01038      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
01039      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
01040      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
01041      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
01042      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
01043      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
01044      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
01045      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
01046      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
01047      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
01048     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
01049      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
01050      {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
01051      {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
01052      {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
01053      {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
01054      {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
01055      {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
01056      {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
01057      {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
01058      {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
01059      {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
01060      {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
01061      {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
01062      {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
01063      {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
01064      {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
01065      {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
01066     {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
01067      {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
01068      {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
01069      {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
01070      {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
01071      {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
01072      {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
01073      {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
01074      {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
01075      {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
01076      {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
01077      {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
01078      {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
01079      {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
01080      {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
01081      {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
01082      {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
01083      {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
01084     {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
01085      {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
01086      {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
01087      {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
01088      {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
01089      {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
01090      {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
```

```
01091       {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
01092       {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
01093       {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
01094       {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
01095       {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
01096       {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
01097       {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
01098       {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
01099       {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
01100       {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
01101       {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
01102      {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
01103       {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
01104       {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
01105       {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
01106       {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
01107       {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
01108       {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
01109       {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
01110       {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
01111       {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
01112       {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
01113       {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
01114       {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
01115       {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
01116       {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
01117       {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
01118       {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
01119       {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
01120      {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
01121       {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
01122       {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
01123       {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
01124       {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
01125       {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
01126       {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
01127       {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
01128       {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
01129       {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
01130       {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
01131       {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
01132       {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
01133       {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
01134       {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
01135       {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
01136       {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
01137       {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
01138      {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
01139       {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
01140       {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
01141       {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
01142       {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
01143       {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
01144       {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
01145       {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
01146       {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
01147       {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
01148       {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
01149       {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
01150       {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
01151       {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
01152       {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
01153       {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
01154       {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
01155       {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
01156      {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
01157       {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
01158       {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
01159       {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
01160       {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
01161       {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
01162       {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
01163       {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
01164       {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
01165       {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
01166       {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
01167       {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
01168       {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
01169       {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
01170       {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
01171       {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
01172       {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
01173       {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
01174      {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
01175       {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
01176       {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
01177       {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
```

```
01178       {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
01179       {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
01180       {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
01181       {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
01182       {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
01183       {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
01184       {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
01185       {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
01186       {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
01187       {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
01188       {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
01189       {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
01190       {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
01191       {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
01192   };
01193
01194   double aux00, aux01, aux10, aux11, sec;
01195
01196   int ilat, ip, isec;
01197
01198   /* Get seconds since begin of year... */
01199   sec = fmod(t, 365.25 * 86400.);
01200
01201   /* Get indices... */
01202   ilat = locate(lats, 18, lat);
01203   ip = locate(ps, 10, p);
01204   isec = locate(secs, 12, sec);
01205
01206   /* Interpolate... */
01207   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
01208               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
01209   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
01210               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
01211   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
01212               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
01213   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
01214               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
01215   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
01216   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
01217   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
01218 }
01219
01220 /*****************************************************************************/
01221
01222 void module_position(
01223   met_t * met0,
01224   met_t * met1,
01225   atm_t * atm,
01226   int ip) {
01227
01228   double ps;
01229
01230   /* Calculate modulo... */
01231   atm->lon[ip] = fmod(atm->lon[ip], 360);
01232   atm->lat[ip] = fmod(atm->lat[ip], 360);
01233
01234   /* Check latitude... */
01235   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01236     if (atm->lat[ip] > 90) {
01237       atm->lat[ip] = 180 - atm->lat[ip];
01238       atm->lon[ip] += 180;
01239     }
01240     if (atm->lat[ip] < -90) {
01241       atm->lat[ip] = -180 - atm->lat[ip];
01242       atm->lon[ip] += 180;
01243     }
01244   }
01245
01246   /* Check longitude... */
01247   while (atm->lon[ip] < -180)
01248     atm->lon[ip] += 360;
01249   while (atm->lon[ip] >= 180)
01250     atm->lon[ip] -= 360;
01251
01252   /* Get surface pressure... */
01253   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01254                   atm->lon[ip], atm->lat[ip], &ps, NULL,
01255                   NULL, NULL, NULL, NULL, NULL);
01256
01257   /* Check pressure... */
01258   if (atm->p[ip] > ps)
01259     atm->p[ip] = ps;
01260   else if (atm->p[ip] < met0->p[met0->np - 1])
01261     atm->p[ip] = met0->p[met0->np - 1];
01262 }
01263
01264 /*****************************************************************************/
```

```
01265
01266 void module_sedi(
01267   ctl_t * ctl,
01268   met_t * met0,
01269   met_t * met1,
01270   atm_t * atm,
01271   int ip,
01272   double dt) {
01273
01274   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
01275   const double A = 1.249, B = 0.42, C = 0.87;
01276
01277   /* Specific gas constant for dry air [J/(kg K)]: */
01278   const double R = 287.058;
01279
01280   /* Average mass of an air molecule [kg/molec]: */
01281   const double m = 4.8096e-26;
01282
01283   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01284
01285   /* Check if parameters are available... */
01286   if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
01287     return;
01288
01289   /* Convert units... */
01290   p = 100 * atm->p[ip];
01291   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01292   rho_p = atm->q[ctl->qnt_rho][ip];
01293
01294   /* Get temperature... */
01295   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
01296                   atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
01297
01298   /* Density of dry air... */
01299   rho = p / (R * T);
01300
01301   /* Dynamic viscosity of air... */
01302   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01303
01304   /* Thermal velocity of an air molecule... */
01305   v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
01306
01307   /* Mean free path of an air molecule... */
01308   lambda = 2 * eta / (rho * v);
01309
01310   /* Knudsen number for air... */
01311   K = lambda / r_p;
01312
01313   /* Cunningham slip-flow correction... */
01314   G = 1 + K * (A + B * exp(-C / K));
01315
01316   /* Sedimentation (fall) velocity... */
01317   v_p =
01318     2. * gsl_pow_2(r_p) * (rho_p -
01319                            rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
01320
01321   /* Calculate pressure change... */
01322   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01323 }
01324
01325 /*****************************************************************************/
01326
01327 void write_output(
01328   const char *dirname,
01329   ctl_t * ctl,
01330   met_t * met0,
01331   met_t * met1,
01332   atm_t * atm,
01333   double t) {
01334
01335   char filename[LEN];
01336
01337   double r;
01338
01339   int year, mon, day, hour, min, sec;
01340
01341   /* Get time... */
01342   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01343
01344   /* Write atmospheric data... */
01345   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01346     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01347             dirname, ctl->atm_basename, year, mon, day, hour, min);
01348     write_atm(filename, ctl, atm, t);
01349   }
01350
```

```
01351   /* Write CSI data... */
01352   if (ctl->csi_basename[0] != '-') {
01353     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01354     write_csi(filename, ctl, atm, t);
01355   }
01356
01357   /* Write ensemble data... */
01358   if (ctl->ens_basename[0] != '-') {
01359     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01360     write_ens(filename, ctl, atm, t);
01361   }
01362
01363   /* Write gridded data... */
01364   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01365     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01366             dirname, ctl->grid_basename, year, mon, day, hour, min);
01367     write_grid(filename, ctl, met0, met1, atm, t);
01368   }
01369
01370   /* Write profile data... */
01371   if (ctl->prof_basename[0] != '-') {
01372     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01373     write_prof(filename, ctl, met0, met1, atm, t);
01374   }
01375
01376   /* Write station data... */
01377   if (ctl->stat_basename[0] != '-') {
01378     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01379     write_station(filename, ctl, atm, t);
01380   }
01381 }
```

5.33 wind.c File Reference

Create meteorological data files with synthetic wind fields.

Functions

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

5.33.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.33.2 Function Documentation

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

Definition at line 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 }
```

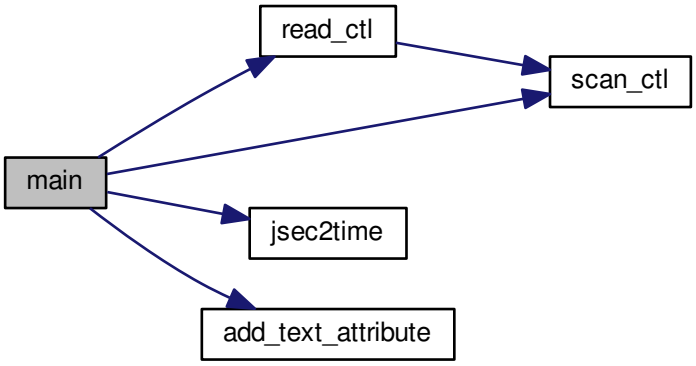
5.33.2.2 int main (int *argc*, char * *argv[]*)

Definition at line 41 of file wind.c.

```
00043                {
00044
00045   ctl_t ctl;
00046
00047   static char filename[LEN];
00048
00049   static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050     u0, u1, alpha;
00051
00052   static float *dataT, *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:



5.34 wind.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* ------------------------------------------------------------
00028    Functions...
00029    ------------------------------------------------------------ */
00030
00031 void add_text_attribute(
00032   int ncid,
00033   char *varname,
00034   char *attrname,
00035   char *text);
00036
00037 /* ------------------------------------------------------------
00038    Main...
00039    ------------------------------------------------------------ */
00040
00041 int main(
00042   int argc,
00043   char *argv[]) {
00044
00045   ctl_t ctl;
00046
00047   static char filename[LEN];
00048
00049   static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050     u0, u1, alpha;
00051
00052   static float *dataT, *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