MPTRAC

Generated by Doxygen 1.8.11

Contents

1 Main Page

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

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 atm_t Struct Reference

Atmospheric data.

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

Data Fields

- int np

  Number of air pacels.
- double time [NP]

  Time [s].
- double p [NP]

  Pressure [hPa].
- double lon [NP]

  Longitude [deg].

- double lat [NP]

  Latitude [deg].
- double q [NQ][NP]

  Quantitiy data (for various, user-defined attributes).
- float up [NP]

  Zonal wind perturbation [m/s].
- float vp [NP]

  Meridional wind perturbation [m/s].
- float wp [NP]

  Vertical velocity perturbation [hPa/s].

4.1.1 Detailed Description

Atmospheric data.

Definition at line 540 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

Definition at line 543 of file libtrac.h.

4.1.2.2 double atm_t::time[NP]

Time [s].

Definition at line 546 of file libtrac.h.

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

Definition at line 549 of file libtrac.h.

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 552 of file libtrac.h.

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 555 of file libtrac.h.

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

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

Definition at line 558 of file libtrac.h.

4.1.2.7 float atm_t::up[NP]

Zonal wind perturbation [m/s].

Definition at line 561 of file libtrac.h.

4.1.2.8 float atm_t::vp[NP]

Meridional wind perturbation [m/s].

Definition at line 564 of file libtrac.h.

4.1.2.9 float atm_t::wp[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 567 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.

Stride for pressure levels.

- int met_np

  Number of target pressure levels.

- double met_p [EP]

  Target pressure levels [hPa].

- int met_tropo

  Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO_1st, 4=WMO_2nd).

- char met_geopot [LEN]

  Surface geopotential data file.

- char met_stage [LEN]

  Command to stage meteo data.

- int isosurf

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

- char balloon [LEN]

  Balloon position filename.

- double turb_dx_trop

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

- double turb_dx_strat

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

- double turb_dz_trop

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

- double turb_dz_strat

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

- double turb_meso

  Scaling factor for mesoscale wind fluctuations.

- double tdec_trop

  Life time of particles (troposphere) [s].

- double tdec_strat

  Life time of particles (stratosphere) [s].

- double psc_h2o

  H2O volume mixing ratio for PSC analysis.

- double psc_hno3

  HNO3 volume mixing ratio for PSC analysis.

- char gw_basename [LEN]

  Basename for gravity wave variance data.

- char atm_basename [LEN]

  Basename of atmospheric data files.

- char atm_gpfile [LEN]

  Gnuplot file for atmospheric data.

- double atm_dt_out

  Time step for atmospheric data output [s].

- int atm_filter

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

- int atm_type

  Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).

- 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 239 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 242 of file libtrac.h.

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

Quantity names.

Definition at line 245 of file libtrac.h.

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

Quantity units.

Definition at line 248 of file libtrac.h.

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

Quantity output format.

Definition at line 251 of file libtrac.h.

4.2.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 254 of file libtrac.h.

4.2.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 257 of file libtrac.h.

4.2.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 260 of file libtrac.h.

4.2.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 263 of file libtrac.h.

4.2.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 266 of file libtrac.h.

4.2.2.10 int ctl_t::qnt_pt

Quantity array index for tropopause pressure.

Definition at line 269 of file libtrac.h.

4.2.2.11 int ctl_t::qnt_z

Quantity array index for geopotential height.

Definition at line 272 of file libtrac.h.

4.2.2.12 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 275 of file libtrac.h.

4.2.2.13 int ctl_t::qnt_t

Quantity array index for temperature.

Definition at line 278 of file libtrac.h.

4.2.2.14 int ctl_t::qnt_u

Quantity array index for zonal wind.

Definition at line 281 of file libtrac.h.

4.2.2.15 int ctl_t::qnt_v

Quantity array index for meridional wind.

Definition at line 284 of file libtrac.h.

4.2.2.16 int ctl_t::qnt_w

Quantity array index for vertical velocity.

Definition at line 287 of file libtrac.h.

4.2.2.17 int ctl_t::qnt_h2o

Quantity array index for water vapor vmr.

Definition at line 290 of file libtrac.h.

4.2.2.18 int ctl_t::qnt_o3

Quantity array index for ozone vmr.

Definition at line 293 of file libtrac.h.

4.2.2.19 int ctl_t::qnt_theta

Quantity array index for potential temperature.

Definition at line 296 of file libtrac.h.

4.2.2.20 int ctl_t::qnt_pv

Quantity array index for potential vorticity.

Definition at line 299 of file libtrac.h.

4.2.2.21 int ctl_t::qnt_tice

Quantity array index for T_ice.

Definition at line 302 of file libtrac.h.

4.2.2.22 int ctl_t::qnt_tsts

Quantity array index for T_STS.

Definition at line 305 of file libtrac.h.

4.2.2.23 int ctl_t::qnt_tnat

Quantity array index for T_NAT.

Definition at line 308 of file libtrac.h.

4.2.2.24 int ctl_t::qnt_stat

Quantity array index for station flag.

Definition at line 311 of file libtrac.h.

4.2.2.25 int ctl_t::qnt_gw_var

Quantity array index for gravity wave variances.

Definition at line 314 of file libtrac.h.

4.2.2.26 int ctl_t::direction

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

Definition at line 317 of file libtrac.h.

4.2.2.27 double ctl_t::t_start

Start time of simulation [s].

Definition at line 320 of file libtrac.h.

4.2.2.28 double ctl_t::t_stop

Stop time of simulation [s].

Definition at line 323 of file libtrac.h.

4.2.2.29 double ctl_t::dt_mod

Time step of simulation [s].

Definition at line 326 of file libtrac.h.

4.2.2.30 double ctl_t::dt_met

Time step of meteorological data [s].

Definition at line 329 of file libtrac.h.

4.2.2.31 int ctl_t::met_dx

Stride for longitudes.

Definition at line 332 of file libtrac.h.

4.2.2.32 int ctl_t::met_dy

Stride for latitudes.

Definition at line 335 of file libtrac.h.

4.2.2.33 int ctl_t::met_dp

Stride for pressure levels.

Definition at line 338 of file libtrac.h.

4.2.2.34 int ctl_t::met_np

Number of target pressure levels.

Definition at line 341 of file libtrac.h.

4.2.2.35 double ctl_t::met_p[EP]

Target pressure levels [hPa].

Definition at line 344 of file libtrac.h.

4.2.2.36 int ctl_t::met_tropo

Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO_1st, 4=WMO_2nd).

Definition at line 348 of file libtrac.h.

4.2.2.37 char ctl_t::met_geopot[LEN]

Surface geopotential data file.

Definition at line 351 of file libtrac.h.

4.2.2.38 char ctl_t::met_stage[LEN]

Command to stage meteo data.

Definition at line 354 of file libtrac.h.

4.2.2.39 int ctl_t::isosurf

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

Definition at line 358 of file libtrac.h.

4.2.2.40 char ctl_t::balloon[LEN]

Balloon position filename.

Definition at line 361 of file libtrac.h.

4.2.2.41 double ctl_t::turb_dx_trop

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

Definition at line 364 of file libtrac.h.

4.2.2.42 double ctl_t::turb_dx_strat

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

Definition at line 367 of file libtrac.h.

4.2.2.43 double ctl_t::turb_dz_trop

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

Definition at line 370 of file libtrac.h.

4.2.2.44 double ctl_t::turb_dz_strat

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

Definition at line 373 of file libtrac.h.

4.2.2.45 double ctl_t::turb_meso

Scaling factor for mesoscale wind fluctuations.

Definition at line 376 of file libtrac.h.

4.2.2.46 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 379 of file libtrac.h.

4.2.2.47 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 382 of file libtrac.h.

4.2.2.48 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 385 of file libtrac.h.

4.2.2.49 double ctl_t::psc_hno3

HNO3 volume mixing ratio for PSC analysis.

Definition at line 388 of file libtrac.h.

4.2.2.50 char ctl_t::gw_basename[LEN]

Basename for gravity wave variance data.

Definition at line 391 of file libtrac.h.

4.2.2.51 char ctl_t::atm_basename[LEN]

Basename of atmospheric data files.

Definition at line 394 of file libtrac.h.

4.2.2.52 char ctl_t::atm_gpfile[LEN]

Gnuplot file for atmospheric data.

Definition at line 397 of file libtrac.h.

4.2.2.53 double ctl_t::atm_dt_out

Time step for atmospheric data output [s].

Definition at line 400 of file libtrac.h.

4.2.2.54 int ctl_t::atm_filter

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

Definition at line 403 of file libtrac.h.

4.2.2.55 int ctl_t::atm_type

Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).

Definition at line 406 of file libtrac.h.

4.2.2.56 char ctl_t::csi_basename[LEN]

Basename of CSI data files.

Definition at line 409 of file libtrac.h.

4.2.2.57 double ctl_t::csi_dt_out

Time step for CSI data output [s].

Definition at line 412 of file libtrac.h.

4.2.2.58 char ctl_t::csi_obsfile[LEN]

Observation data file for CSI analysis.

Definition at line 415 of file libtrac.h.

4.2.2.59 double ctl_t::csi_obsmin

Minimum observation index to trigger detection.

Definition at line 418 of file libtrac.h.

4.2.2.60 double ctl_t::csi_modmin

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

Definition at line 421 of file libtrac.h.

4.2.2.61 int ctl_t::csi_nz

Number of altitudes of gridded CSI data.

Definition at line 424 of file libtrac.h.

4.2.2.62 double ctl_t::csi_z0

Lower altitude of gridded CSI data [km].

Definition at line 427 of file libtrac.h.

4.2.2.63 double ctl_t::csi_z1

Upper altitude of gridded CSI data [km].

Definition at line 430 of file libtrac.h.

4.2.2.64 int ctl_t::csi_nx

Number of longitudes of gridded CSI data.

Definition at line 433 of file libtrac.h.

4.2.2.65 double ctl_t::csi_lon0

Lower longitude of gridded CSI data [deg].

Definition at line 436 of file libtrac.h.

4.2.2.66 double ctl_t::csi_lon1

Upper longitude of gridded CSI data [deg].

Definition at line 439 of file libtrac.h.

4.2.2.67 int ctl_t::csi_ny

Number of latitudes of gridded CSI data.

Definition at line 442 of file libtrac.h.

4.2.2.68 double ctl_t::csi_lat0

Lower latitude of gridded CSI data [deg].

Definition at line 445 of file libtrac.h.

4.2.2.69 double ctl_t::csi_lat1

Upper latitude of gridded CSI data [deg].

Definition at line 448 of file libtrac.h.

4.2.2.70 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 451 of file libtrac.h.

4.2.2.71 char ctl_t::grid_gpfile[LEN]

Gnuplot file for gridded data.

Definition at line 454 of file libtrac.h.

4.2.2.72 double ctl_t::grid_dt_out

Time step for gridded data output [s].

Definition at line 457 of file libtrac.h.

4.2.2.73 int ctl_t::grid_sparse

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

Definition at line 460 of file libtrac.h.

4.2.2.74 int ctl_t::grid_nz

Number of altitudes of gridded data.

Definition at line 463 of file libtrac.h.

4.2.2.75 double ctl_t::grid_z0

Lower altitude of gridded data [km].

Definition at line 466 of file libtrac.h.

4.2.2.76 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 469 of file libtrac.h.

4.2.2.77 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 472 of file libtrac.h.

4.2.2.78 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 475 of file libtrac.h.

4.2.2.79 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 478 of file libtrac.h.

4.2.2.80 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 481 of file libtrac.h.

4.2.2.81 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 484 of file libtrac.h.

4.2.2.82 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 487 of file libtrac.h.

4.2.2.83 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 490 of file libtrac.h.

4.2.2.84 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 493 of file libtrac.h.

4.2.2.85 int ctl_t::prof_nz

Number of altitudes of gridded profile data.

Definition at line 496 of file libtrac.h.

4.2.2.86 double ctl_t::prof_z0

Lower altitude of gridded profile data [km].

Definition at line 499 of file libtrac.h.

4.2.2.87 double ctl_t::prof_z1

Upper altitude of gridded profile data [km].

Definition at line 502 of file libtrac.h.

4.2.2.88 int ctl_t::prof_nx

Number of longitudes of gridded profile data.

Definition at line 505 of file libtrac.h.

4.2.2.89 double ctl_t::prof_lon0

Lower longitude of gridded profile data [deg].

Definition at line 508 of file libtrac.h.

4.2.2.90 double ctl_t::prof_lon1

Upper longitude of gridded profile data [deg].

Definition at line 511 of file libtrac.h.

4.2.2.91 int ctl_t::prof_ny

Number of latitudes of gridded profile data.

Definition at line 514 of file libtrac.h.

4.2.2.92 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 517 of file libtrac.h.

4.2.2.93 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 520 of file libtrac.h.

4.2.2.94 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 523 of file libtrac.h.

4.2.2.95 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 526 of file libtrac.h.

4.2.2.96 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 529 of file libtrac.h.

4.2.2.97 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 532 of file libtrac.h.

4.2.2.98 double ctl_t::stat_r

Search radius around station [km].

Definition at line 535 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].
- double pt [EX][EY]
  Tropopause pressure [hPa].
- float pl [EX][EY][EP]
  Pressure on model levels [hPa].
- float z [EX][EY][EP]
  Geopotential height [km].
- 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 572 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 double met_t::time

Time [s].

Definition at line 575 of file libtrac.h.

4.3.2.2 int met_t::nx

Number of longitudes.

Definition at line 578 of file libtrac.h.

4.3.2.3 int met_t::ny

Number of latitudes.

Definition at line 581 of file libtrac.h.

4.3.2.4 int met_t::np

Number of pressure levels.

Definition at line 584 of file libtrac.h.

4.3.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 587 of file libtrac.h.

4.3.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 590 of file libtrac.h.

4.3.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 593 of file libtrac.h.

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

Surface pressure [hPa].

Definition at line 596 of file libtrac.h.

4.3.2.9 double met_t::pt[EX][EY]

Tropopause pressure [hPa].

Definition at line 599 of file libtrac.h.

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

Pressure on model levels [hPa].

Definition at line 602 of file libtrac.h.

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

Geopotential height [km].

Definition at line 605 of file libtrac.h.

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

Temperature [K].

Definition at line 608 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 611 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 614 of file libtrac.h.

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

Vertical wind [hPa/s].

Definition at line 617 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 620 of file libtrac.h.

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

Ozone volume mixing ratio [1].

Definition at line 623 of file libtrac.h.

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

- libtrac.h

5 File Documentation

5.1 center.c File Reference

Calculate center of mass of air parcels.

Functions

- int main (int argc, char ∗argv[ ])

5.1.1 Detailed Description

Calculate center of mass of air parcels.

Definition in file center.c.

5.1.2 Function Documentation

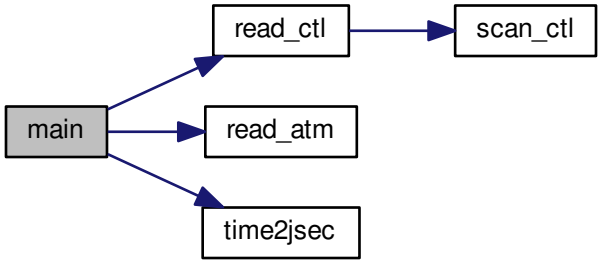
5.1.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 27 of file center.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double latm, lats, lonm, lons, t, zm, zs;
00040
00041   int f, ip, year, mon, day, hour, min;
00042
00043   /* Allocate... */
00044   ALLOC(atm, atm_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052
00053   /* Write info... */
00054   printf("Write center of mass data: %s\n", argv[2]);
00055
00056   /* Create output file... */
00057   if (!(out = fopen(argv[2], "w")))
00058     ERRMSG("Cannot create file!");
00059
00060   /* Write header... */
00061   fprintf(out,
00062           "# $1  = time [s]\n"
00063           "# $2  = altitude (mean) [km]\n"
00064           "# $3  = altitude (sigma) [km]\n"
00065           "# $4  = altitude (minimum) [km]\n"
00066           "# $5  = altitude (10%% percentile) [km]\n"
00067           "# $6  = altitude (1st quarter) [km]\n"
00068           "# $7  = altitude (median) [km]\n"
00069           "# $8  = altitude (3rd quarter) [km]\n"
00070           "# $9  = altitude (90%% percentile) [km]\n"
00071           "# $10 = altitude (maximum) [km]\n");
00072   fprintf(out,
00073           "# $11 = longitude (mean) [deg]\n"
00074           "# $12 = longitude (sigma) [deg]\n"
00075           "# $13 = longitude (minimum) [deg]\n"
00076           "# $14 = longitude (10%% percentile) [deg]\n"
00077           "# $15 = longitude (1st quarter) [deg]\n"
00078           "# $16 = longitude (median) [deg]\n"
00079           "# $17 = longitude (3rd quarter) [deg]\n"
00080           "# $18 = longitude (90%% percentile) [deg]\n"
00081           "# $19 = longitude (maximum) [deg]\n");
00082   fprintf(out,
00083           "# $20 = latitude (mean) [deg]\n"
00084           "# $21 = latitude (sigma) [deg]\n"
00085           "# $22 = latitude (minimum) [deg]\n"
```

```
00086           "# $23 = latitude (10%% percentile) [deg]\n"
00087           "# $24 = latitude (1st quarter) [deg]\n"
00088           "# $25 = latitude (median) [deg]\n"
00089           "# $26 = latitude (3rd quarter) [deg]\n"
00090           "# $27 = latitude (90%% percentile) [deg]\n"
00091           "# $28 = latitude (maximum) [deg]\n\n");
00092
00093   /* Loop over files... */
00094   for (f = 3; f < argc; f++) {
00095
00096     /* Read atmopheric data... */
00097     read_atm(argv[f], &ctl, atm);
00098
00099     /* Initialize... */
00100     zm = zs = 0;
00101     lonm = lons = 0;
00102     latm = lats = 0;
00103
00104     /* Calculate mean and standard deviation... */
00105     for (ip = 0; ip < atm->np; ip++) {
00106       zm += Z(atm->p[ip]) / atm->np;
00107       lonm += atm->lon[ip] / atm->np;
00108       latm += atm->lat[ip] / atm->np;
00109       zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00110       lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00111       lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00112     }
00113
00114     /* Normalize... */
00115     zs = sqrt(zs - gsl_pow_2(zm));
00116     lons = sqrt(lons - gsl_pow_2(lonm));
00117     lats = sqrt(lats - gsl_pow_2(latm));
00118
00119     /* Sort arrays... */
00120     gsl_sort(atm->p, 1, (size_t) atm->np);
00121     gsl_sort(atm->lon, 1, (size_t) atm->np);
00122     gsl_sort(atm->lat, 1, (size_t) atm->np);
00123
00124     /* Get time from filename... */
00125     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00126     year = atoi(tstr);
00127     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00128     mon = atoi(tstr);
00129     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00130     day = atoi(tstr);
00131     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00132     hour = atoi(tstr);
00133     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00134     min = atoi(tstr);
00135     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00136
00137     /* Write data... */
00138     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g "
00139             "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00140             t, zm, zs, Z(atm->p[atm->np - 1]),
00141             Z(atm->p[atm->np - atm->np / 10]),
00142             Z(atm->p[atm->np - atm->np / 4]),
00143             Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
00144             Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00145             lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00146             atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00147             atm->lon[atm->np - atm->np / 4],
00148             atm->lon[atm->np - atm->np / 10],
00149             atm->lon[atm->np - 1],
00150             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00151             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00152             atm->lat[atm->np - atm->np / 4],
00153             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00154   }
00155
00156   /* Close file... */
00157   fclose(out);
00158
00159   /* Free... */
00160   free(atm);
00161
00162   return EXIT_SUCCESS;
00163 }
```

Here is the call graph for this function:



5.2 center.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double latm, lats, lonm, lons, t, zm, zs;
00040
00041   int f, ip, year, mon, day, hour, min;
00042
00043   /* Allocate... */
00044   ALLOC(atm, atm_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052
00053   /* Write info... */
00054   printf("Write center of mass data: %s\n", argv[2]);
00055
00056   /* Create output file... */
00057   if (!(out = fopen(argv[2], "w")))
00058     ERRMSG("Cannot create file!");
00059
```

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

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

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

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

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

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

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

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

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

```
00147             atm->lon[atm->np - atm->np / 4],
00148             atm->lon[atm->np - atm->np / 10],
00149             atm->lon[atm->np - 1],
00150             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00151             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00152             atm->lat[atm->np - atm->np / 4],
00153             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00154   }
00155
00156   /* Close file... */
00157   fclose(out);
00158
00159   /* Free... */
00160   free(atm);
00161
00162   return EXIT_SUCCESS;
00163 }
```

5.3 cluster.c File Reference

Clustering of trajectories.

Functions

- int main (int argc, char ∗argv[ ])

5.3.1 Detailed Description

Clustering of trajectories.

Definition in file cluster.c.

5.3.2 Function Documentation

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

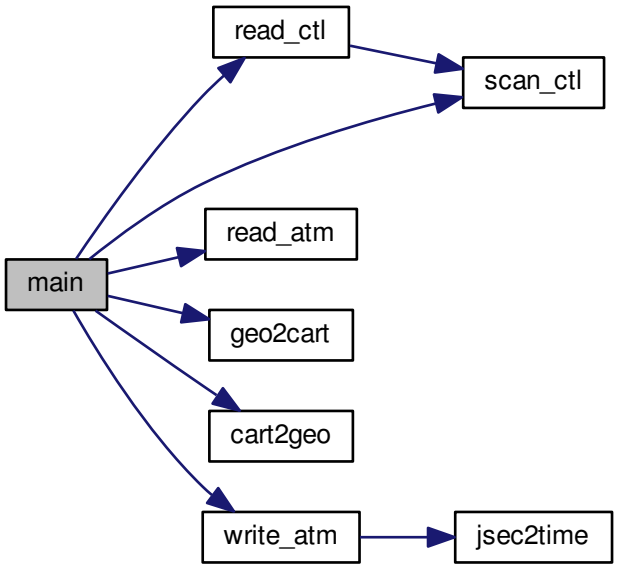
Definition at line 41 of file cluster.c.

```
00043                {
00044
00045   ctl_t ctl;
00046
00047   atm_t *atm;
00048
00049   gsl_rng *rng;
00050
00051   FILE *out;
00052
00053   static double d2, *dist, lat, lon, rmsd[NS],
00054     x[3], xs[NT][NS][3], z, zs[NT][NS];
00055
00056   static int *cluster, f, idx[NS], ip, is, it, itmax, np[NS], ns;
00057
00058   /* Check arguments... */
00059   if (argc < 4)
00060     ERRMSG("Give parameters: <ctl> <cluster.log> <atm1> [<atm2> <atm3> ...]");
00061
00062   /* Read control parameters... */
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   ns = (int) scan_ctl(argv[1], argc, argv, "CLUSTER_NS", -1, "7", NULL);
00065   if (ns > NS)
00066     ERRMSG("Too many seeds!");
00067   itmax =
00068     (int) scan_ctl(argv[1], argc, argv, "CLUSTER_ITMAX", -1, "10", NULL);
00069
```

```
00070   /* Initialize random number generator... */
00071   gsl_rng_env_setup();
00072   rng = gsl_rng_alloc(gsl_rng_default);
00073
00074   /* Allocate... */
00075   ALLOC(atm, atm_t, 1);
00076   ALLOC(cluster, int,
00077         NP);
00078   ALLOC(dist, double,
00079         NP * NS);
00080
00081   /* Create output file... */
00082   printf("Write cluster data: %s\n", argv[2]);
00083   if (!(out = fopen(argv[2], "w")))
00084     ERRMSG("Cannot create file!");
00085
00086   /* Write header... */
00087   fprintf(out,
00088           "# $1 = iteration index\n"
00089           "# $2 = seed index\n"
00090           "# $3 = time step index\n"
00091           "# $4 = mean altitude [km]\n"
00092           "# $5 = mean longitude [deg]\n"
00093           "# $6 = mean latitude [deg]\n"
00094           "# $7 = number of points\n" "# $8 = RMSD [km^2]\n");
00095
00096   /* Get seeds (random selection of trajectories)... */
00097   for (f = 3; f < argc; f++) {
00098
00099     /* Check number of timesteps... */
00100     if (f - 3 > NT)
00101       ERRMSG("Too many timesteps!");
00102
00103     /* Read atmopheric data... */
00104     read_atm(argv[f], &ctl, atm);
00105
00106     /* Pick seeds (random selection)... */
00107     if (f == 3)
00108       for (is = 0; is < ns; is++)
00109         idx[is] = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00110
00111     /* Save seeds... */
00112     for (is = 0; is < ns; is++) {
00113       geo2cart(0, atm->lon[idx[is]], atm->lat[idx[is]], xs[f - 3][is]);
00114       zs[f - 3][is] = Z(atm->p[idx[is]]);
00115     }
00116   }
00117
00118   /* Iterations... */
00119   for (it = 0; it < itmax; it++) {
00120
00121     /* Write output... */
00122     for (is = 0; is < ns; is++) {
00123       fprintf(out, "\n");
00124       for (f = 3; f < argc; f++) {
00125         cart2geo(xs[f - 3][is], &z, &lon, &lat);
00126         fprintf(out, "%d %d %d %g %g %g %d %g\n",
00127                 it, is, f - 3, zs[f - 3][is], lon, lat, np[is], rmsd[is]);
00128       }
00129     }
00130
00131     /* Init... */
00132     for (ip = 0; ip < atm->np; ip++)
00133       for (is = 0; is < ns; is++) {
00134         dist[ip * NS + is] = 0;
00135         rmsd[is] = 0;
00136       }
00137
00138     /* Get distances between seeds and trajectories... */
00139     for (f = 3; f < argc; f++) {
00140
00141       /* Read atmopheric data... */
00142       read_atm(argv[f], &ctl, atm);
00143
00144       /* Get distances... */
00145       for (ip = 0; ip < atm->np; ip++) {
00146         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00147         z = Z(atm->p[ip]);
00148         for (is = 0; is < ns; is++) {
00149           d2 =
00150             DIST2(x, xs[f - 3][is]) + gsl_pow_2((z - zs[f - 3][is]) * 200.);
00151           dist[ip * NS + is] += d2;
00152           rmsd[is] += d2;
00153         }
00154       }
00155     }
00156
```

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

Here is the call graph for this function:



5.4 cluster.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* ------------------------------------------------------------
00028    Defines...
00029    ------------------------------------------------------------ */
00030
00032 #define NS 100
00033
00035 #define NT 1000
00036
00037 /* ------------------------------------------------------------
00038    Main...
00039    ------------------------------------------------------------ */
00040
00041 int main(
00042   int argc,
00043   char *argv[]) {
00044
00045   ctl_t ctl;
00046
```

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

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

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

5.5 conv.c File Reference

Convert file format of atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.5.1 Detailed Description

Convert file format of atmospheric data files.

Definition in file conv.c.

5.5.2 Function Documentation

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

Definition at line 27 of file conv.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   /* Check arguments... */
00036   if (argc < 6)
00037     ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038            " <atm_out> <atm_out_type>");
00039
00040   /* Allocate... */
00041   ALLOC(atm, atm_t, 1);
00042
00043   /* Read control parameters... */
00044   read_ctl(argv[1], argc, argv, &ctl);
00045
00046   /* Read atmospheric data... */
00047   ctl.atm_type = atoi(argv[3]);
00048   read_atm(argv[2], &ctl, atm);
00049
00050   /* Write atmospheric data... */
00051   ctl.atm_type = atoi(argv[5]);
00052   write_atm(argv[4], &ctl, atm, 0);
00053
00054   /* Free... */
00055   free(atm);
00056
00057   return EXIT_SUCCESS;
00058 }
```

Here is the call graph for this function:

5.6 conv.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   atm_t *atm;
00034
00035   /* Check arguments... */
00036   if (argc < 6)
00037     ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038            " <atm_out> <atm_out_type>");
00039
00040   /* Allocate... */
00041   ALLOC(atm, atm_t, 1);
00042
00043   /* Read control parameters... */
00044   read_ctl(argv[1], argc, argv, &ctl);
00045
00046   /* Read atmospheric data... */
00047   ctl.atm_type = atoi(argv[3]);
00048   read_atm(argv[2], &ctl, atm);
00049
00050   /* Write atmospheric data... */
00051   ctl.atm_type = atoi(argv[5]);
00052   write_atm(argv[4], &ctl, atm, 0);
00053
00054   /* Free... */
00055   free(atm);
00056
00057   return EXIT_SUCCESS;
00058 }
```

5.7 dist.c File Reference

Calculate transport deviations of trajectories.

Functions

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

5.7.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

5.7.2 Function Documentation

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

Definition at line 27 of file dist.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double ahtd, aqtd[NQ], atce1[NQ], atce2[NQ], avtd, lat0, lat1,
00040     *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041     *lv1, *lv2, p0, p1, *q1, *q2, rhtd, rqtd[NQ], rtce1[NQ], rtce2[NQ], rvtd,
00042     t, t0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old;
00043
00044   int ens, f, ip, iq, np, year, mon, day, hour, min;
00045
00046   /* Allocate... */
00047   ALLOC(atm1, atm_t, 1);
00048   ALLOC(atm2, atm_t, 1);
00049   ALLOC(lon1_old, double,
00050         NP);
00051   ALLOC(lat1_old, double,
00052         NP);
00053   ALLOC(z1_old, double,
00054         NP);
00055   ALLOC(lh1, double,
00056         NP);
00057   ALLOC(lv1, double,
00058         NP);
00059   ALLOC(lon2_old, double,
00060         NP);
00061   ALLOC(lat2_old, double,
00062         NP);
00063   ALLOC(z2_old, double,
00064         NP);
00065   ALLOC(lh2, double,
00066         NP);
00067   ALLOC(lv2, double,
00068         NP);
00069   ALLOC(q1, double,
00070         NQ * NP);
00071   ALLOC(q2, double,
00072         NQ * NP);
00073
00074   /* Check arguments... */
00075   if (argc < 5)
00076     ERRMSG("Give parameters: <ctl> <outfile> <atm1a> <atm1b>"
00077            " [<atm2a> <atm2b> ...]");
00078
00079   /* Read control parameters... */
00080   read_ctl(argv[1], argc, argv, &ctl);
00081   ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
00082   p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00083   p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00084   lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00085   lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00086   lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00087   lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00088
00089   /* Write info... */
00090   printf("Write transport deviations: %s\n", argv[2]);
00091
00092   /* Create output file... */
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 = trajectory time [s]\n"
00100           "# $3 = AHTD [km]\n"
00101           "# $4 = RHTD [%%]\n" "# $5 = AVTD [km]\n" "# $6 = RVTD [%%]\n");
00102   for (iq = 0; iq < ctl.nq; iq++)
00103     fprintf(out,
00104             "# $%d = AQTD (%s) [%s]\n"
```

```
00105             "# $%d = RQTD (%s) [%%]\n",
00106             7 + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00107             8 + 2 * iq, ctl.qnt_name[iq]);
00108   for (iq = 0; iq < ctl.nq; iq++)
00109     fprintf(out,
00110             "# $%d = ATCE_1 (%s) [%s]\n"
00111             "# $%d = RTCE_1 (%s) [%%]\n",
00112             7 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00113             8 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00114   for (iq = 0; iq < ctl.nq; iq++)
00115     fprintf(out,
00116             "# $%d = ATCE_2 (%s) [%s]\n"
00117             "# $%d = RTCE_2 (%s) [%%]\n",
00118             7 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00119             8 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00120   fprintf(out, "# $%d = number of particles\n\n", 7 + 6 * ctl.nq);
00121
00122   /* Loop over file pairs... */
00123   for (f = 3; f < argc; f += 2) {
00124
00125     /* Read atmopheric data... */
00126     read_atm(argv[f], &ctl, atm1);
00127     read_atm(argv[f + 1], &ctl, atm2);
00128
00129     /* Check if structs match... */
00130     if (atm1->np != atm2->np)
00131       ERRMSG("Different numbers of parcels!");
00132     for (ip = 0; ip < atm1->np; ip++)
00133       if (gsl_finite(atm1->time[ip]) && gsl_finite(atm2->time[ip])
00134           && atm1->time[ip] != atm2->time[ip])
00135         ERRMSG("Times do not match!");
00136
00137     /* Get time from filename... */
00138     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00139     year = atoi(tstr);
00140     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00141     mon = atoi(tstr);
00142     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00143     day = atoi(tstr);
00144     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00145     hour = atoi(tstr);
00146     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00147     min = atoi(tstr);
00148     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00149
00150     /* Save initial data... */
00151     if (f == 3) {
00152       t0 = t;
00153       for (iq = 0; iq < ctl.nq; iq++)
00154         for (ip = 0; ip < atm1->np; ip++) {
00155           q1[iq * NP + ip] = atm1->q[iq][ip];
00156           q2[iq * NP + ip] = atm2->q[iq][ip];
00157         }
00158     }
00159
00160     /* Init... */
00161     np = 0;
00162     ahtd = avtd = rhtd = rvtd = 0;
00163     for (iq = 0; iq < ctl.nq; iq++)
00164       aqtd[iq] = atce1[iq] = atce2[iq] = rqtd[iq] = rtce1[iq] = rtce2[iq] = 0;
00165
00166     /* Loop over air parcels... */
00167     for (ip = 0; ip < atm1->np; ip++) {
00168
00169       /* Check data... */
00170       if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00171         continue;
00172
00173       /* Check ensemble ID... */
00174       if (ens >= 0 && ctl.qnt_ens >= 0 && atm1->q[ctl.qnt_ens][ip] != ens)
00175         continue;
00176       if (ens >= 0 && ctl.qnt_ens >= 0 && atm2->q[ctl.qnt_ens][ip] != ens)
00177         continue;
00178
00179       /* Check spatial range... */
00180       if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00181           || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00182           || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00183         continue;
00184       if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00185           || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00186           || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00187         continue;
00188
00189       /* Convert coordinates... */
00190       geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00191       geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
```

```
00192       z1 = Z(atm1->p[ip]);
00193       z2 = Z(atm2->p[ip]);
00194
00195       /* Calculate absolute transport deviations... */
00196       ahtd += DIST(x1, x2);
00197       avtd += fabs(z1 - z2);
00198       for (iq = 0; iq < ctl.nq; iq++)
00199         aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00200
00201       /* Calculate relative transport deviations... */
00202       if (f > 3) {
00203
00204         /* Get trajectory lengths... */
00205         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00206         lh1[ip] += DIST(x0, x1);
00207         lv1[ip] += fabs(z1_old[ip] - z1);
00208
00209         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00210         lh2[ip] += DIST(x0, x2);
00211         lv2[ip] += fabs(z2_old[ip] - z2);
00212
00213         /* Get relative transport deviations... */
00214         if (lh1[ip] + lh2[ip] > 0)
00215           rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00216         if (lv1[ip] + lv2[ip] > 0)
00217           rvtd += 200. * fabs(z1 - z2) / (lv1[ip] + lv2[ip]);
00218         for (iq = 0; iq < ctl.nq; iq++)
00219           rqtd[iq] += 200. * fabs(atm1->q[iq][ip] - atm2->q[iq][ip])
00220             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00221
00222         /* Get tracer conservation errors... */
00223         for (iq = 0; iq < ctl.nq; iq++) {
00224           atce1[iq] += fabs(atm1->q[iq][ip] - q1[iq * NP + ip]);
00225           rtce1[iq] += 200. * fabs(atm1->q[iq][ip] - q1[iq * NP + ip])
00226             / (fabs(atm1->q[iq][ip]) + fabs(q1[iq * NP + ip]));
00227           atce2[iq] += fabs(atm2->q[iq][ip] - q2[iq * NP + ip]);
00228           rtce2[iq] += 200. * fabs(atm2->q[iq][ip] - q2[iq * NP + ip])
00229             / (fabs(atm2->q[iq][ip]) + fabs(q2[iq * NP + ip]));
00230         }
00231       }
00232
00233       /* Save positions of air parcels... */
00234       lon1_old[ip] = atm1->lon[ip];
00235       lat1_old[ip] = atm1->lat[ip];
00236       z1_old[ip] = z1;
00237
00238       lon2_old[ip] = atm2->lon[ip];
00239       lat2_old[ip] = atm2->lat[ip];
00240       z2_old[ip] = z2;
00241
00242       /* Increment air parcel counter... */
00243       np++;
00244     }
00245
00246     /* Write output... */
00247     fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00248             ahtd / np, rhtd / np, avtd / np, rvtd / np);
00249     for (iq = 0; iq < ctl.nq; iq++) {
00250       fprintf(out, " ");
00251       fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
00252       fprintf(out, " ");
00253       fprintf(out, ctl.qnt_format[iq], rqtd[iq] / np);
00254     }
00255     for (iq = 0; iq < ctl.nq; iq++) {
00256       fprintf(out, " ");
00257       fprintf(out, ctl.qnt_format[iq], atce1[iq] / np);
00258       fprintf(out, " ");
00259       fprintf(out, ctl.qnt_format[iq], rtce1[iq] / np);
00260     }
00261     for (iq = 0; iq < ctl.nq; iq++) {
00262       fprintf(out, " ");
00263       fprintf(out, ctl.qnt_format[iq], atce2[iq] / np);
00264       fprintf(out, " ");
00265       fprintf(out, ctl.qnt_format[iq], rtce2[iq] / np);
00266     }
00267     fprintf(out, " %d\n", np);
00268   }
00269
00270   /* Close file... */
00271   fclose(out);
00272
00273   /* Free... */
00274   free(atm1);
00275   free(atm2);
00276   free(lon1_old);
00277   free(lat1_old);
00278   free(z1_old);
```

```
00279   free(lh1);
00280   free(lv1);
00281   free(lon2_old);
00282   free(lat2_old);
00283   free(z2_old);
00284   free(lh2);
00285   free(lv2);
00286
00287   return EXIT_SUCCESS;
00288 }
```

Here is the call graph for this function:

5.8 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) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double ahtd, aqtd[NQ], atce1[NQ], atce2[NQ], avtd, lat0, lat1,
00040     *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041     *lv1, *lv2, p0, p1, *q1, *q2, rhtd, rqtd[NQ], rtce1[NQ], rtce2[NQ], rvtd,
00042     t, t0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old;
00043
00044   int ens, f, ip, iq, np, year, mon, day, hour, min;
00045
00046   /* Allocate... */
00047   ALLOC(atm1, atm_t, 1);
00048   ALLOC(atm2, atm_t, 1);
00049   ALLOC(lon1_old, double,
00050         NP);
00051   ALLOC(lat1_old, double,
00052         NP);
00053   ALLOC(z1_old, double,
00054         NP);
00055   ALLOC(lh1, double,
00056         NP);
00057   ALLOC(lv1, double,
00058         NP);
00059   ALLOC(lon2_old, double,
00060         NP);
00061   ALLOC(lat2_old, double,
00062         NP);
00063   ALLOC(z2_old, double,
00064         NP);
00065   ALLOC(lh2, double,
00066         NP);
00067   ALLOC(lv2, double,
00068         NP);
00069   ALLOC(q1, double,
00070         NQ * NP);
00071   ALLOC(q2, double,
```

```
00072         NQ * NP);
00073
00074   /* Check arguments... */
00075   if (argc < 5)
00076     ERRMSG("Give parameters: <ctl> <outfile> <atm1a> <atm1b>"
00077            " [<atm2a> <atm2b> ...]");
00078
00079   /* Read control parameters... */
00080   read_ctl(argv[1], argc, argv, &ctl);
00081   ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
00082   p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00083   p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00084   lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00085   lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00086   lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00087   lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00088
00089   /* Write info... */
00090   printf("Write transport deviations: %s\n", argv[2]);
00091
00092   /* Create output file... */
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 = trajectory time [s]\n"
00100           "# $3 = AHTD [km]\n"
00101           "# $4 = RHTD [%%]\n" "# $5 = AVTD [km]\n" "# $6 = RVTD [%%]\n");
00102   for (iq = 0; iq < ctl.nq; iq++)
00103     fprintf(out,
00104             "# $%d = AQTD (%s) [%s]\n"
00105             "# $%d = RQTD (%s) [%%]\n",
00106             7 + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00107             8 + 2 * iq, ctl.qnt_name[iq]);
00108   for (iq = 0; iq < ctl.nq; iq++)
00109     fprintf(out,
00110             "# $%d = ATCE_1 (%s) [%s]\n"
00111             "# $%d = RTCE_1 (%s) [%%]\n",
00112             7 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00113             8 + 2 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00114   for (iq = 0; iq < ctl.nq; iq++)
00115     fprintf(out,
00116             "# $%d = ATCE_2 (%s) [%s]\n"
00117             "# $%d = RTCE_2 (%s) [%%]\n",
00118             7 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq], ctl.qnt_unit[iq],
00119             8 + 4 * ctl.nq + 2 * iq, ctl.qnt_name[iq]);
00120   fprintf(out, "# $%d = number of particles\n\n", 7 + 6 * ctl.nq);
00121
00122   /* Loop over file pairs... */
00123   for (f = 3; f < argc; f += 2) {
00124
00125     /* Read atmopheric data... */
00126     read_atm(argv[f], &ctl, atm1);
00127     read_atm(argv[f + 1], &ctl, atm2);
00128
00129     /* Check if structs match... */
00130     if (atm1->np != atm2->np)
00131       ERRMSG("Different numbers of parcels!");
00132     for (ip = 0; ip < atm1->np; ip++)
00133       if (gsl_finite(atm1->time[ip]) && gsl_finite(atm2->time[ip])
00134           && atm1->time[ip] != atm2->time[ip])
00135         ERRMSG("Times do not match!");
00136
00137     /* Get time from filename... */
00138     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00139     year = atoi(tstr);
00140     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00141     mon = atoi(tstr);
00142     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00143     day = atoi(tstr);
00144     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00145     hour = atoi(tstr);
00146     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00147     min = atoi(tstr);
00148     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00149
00150     /* Save initial data... */
00151     if (f == 3) {
00152       t0 = t;
00153       for (iq = 0; iq < ctl.nq; iq++)
00154         for (ip = 0; ip < atm1->np; ip++) {
00155           q1[iq * NP + ip] = atm1->q[iq][ip];
00156           q2[iq * NP + ip] = atm2->q[iq][ip];
00157         }
00158     }
```

```
00159
00160     /* Init... */
00161     np = 0;
00162     ahtd = avtd = rhtd = rvtd = 0;
00163     for (iq = 0; iq < ctl.nq; iq++)
00164       aqtd[iq] = atce1[iq] = atce2[iq] = rqtd[iq] = rtce1[iq] = rtce2[iq] = 0;
00165
00166     /* Loop over air parcels... */
00167     for (ip = 0; ip < atm1->np; ip++) {
00168
00169       /* Check data... */
00170       if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00171         continue;
00172
00173       /* Check ensemble ID... */
00174       if (ens >= 0 && ctl.qnt_ens >= 0 && atm1->q[ctl.qnt_ens][ip] != ens)
00175         continue;
00176       if (ens >= 0 && ctl.qnt_ens >= 0 && atm2->q[ctl.qnt_ens][ip] != ens)
00177         continue;
00178
00179       /* Check spatial range... */
00180       if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00181           || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00182           || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00183         continue;
00184       if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00185           || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00186           || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00187         continue;
00188
00189       /* Convert coordinates... */
00190       geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00191       geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00192       z1 = Z(atm1->p[ip]);
00193       z2 = Z(atm2->p[ip]);
00194
00195       /* Calculate absolute transport deviations... */
00196       ahtd += DIST(x1, x2);
00197       avtd += fabs(z1 - z2);
00198       for (iq = 0; iq < ctl.nq; iq++)
00199         aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00200
00201       /* Calculate relative transport deviations... */
00202       if (f > 3) {
00203
00204         /* Get trajectory lengths... */
00205         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00206         lh1[ip] += DIST(x0, x1);
00207         lv1[ip] += fabs(z1_old[ip] - z1);
00208
00209         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00210         lh2[ip] += DIST(x0, x2);
00211         lv2[ip] += fabs(z2_old[ip] - z2);
00212
00213         /* Get relative transport deviations... */
00214         if (lh1[ip] + lh2[ip] > 0)
00215           rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00216         if (lv1[ip] + lv2[ip] > 0)
00217           rvtd += 200. * fabs(z1 - z2) / (lv1[ip] + lv2[ip]);
00218         for (iq = 0; iq < ctl.nq; iq++)
00219           rqtd[iq] += 200. * fabs(atm1->q[iq][ip] - atm2->q[iq][ip])
00220             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00221
00222         /* Get tracer conservation errors... */
00223         for (iq = 0; iq < ctl.nq; iq++) {
00224           atce1[iq] += fabs(atm1->q[iq][ip] - q1[iq * NP + ip]);
00225           rtce1[iq] += 200. * fabs(atm1->q[iq][ip] - q1[iq * NP + ip])
00226             / (fabs(atm1->q[iq][ip]) + fabs(q1[iq * NP + ip]));
00227           atce2[iq] += fabs(atm2->q[iq][ip] - q2[iq * NP + ip]);
00228           rtce2[iq] += 200. * fabs(atm2->q[iq][ip] - q2[iq * NP + ip])
00229             / (fabs(atm2->q[iq][ip]) + fabs(q2[iq * NP + ip]));
00230         }
00231       }
00232
00233       /* Save positions of air parcels... */
00234       lon1_old[ip] = atm1->lon[ip];
00235       lat1_old[ip] = atm1->lat[ip];
00236       z1_old[ip] = z1;
00237
00238       lon2_old[ip] = atm2->lon[ip];
00239       lat2_old[ip] = atm2->lat[ip];
00240       z2_old[ip] = z2;
00241
00242       /* Increment air parcel counter... */
00243       np++;
00244     }
00245
```

```
00246     /* Write output... */
00247     fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00248             ahtd / np, rhtd / np, avtd / np, rvtd / np);
00249     for (iq = 0; iq < ctl.nq; iq++) {
00250       fprintf(out, " ");
00251       fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
00252       fprintf(out, " ");
00253       fprintf(out, ctl.qnt_format[iq], rqtd[iq] / np);
00254     }
00255     for (iq = 0; iq < ctl.nq; iq++) {
00256       fprintf(out, " ");
00257       fprintf(out, ctl.qnt_format[iq], atce1[iq] / np);
00258       fprintf(out, " ");
00259       fprintf(out, ctl.qnt_format[iq], rtce1[iq] / np);
00260     }
00261     for (iq = 0; iq < ctl.nq; iq++) {
00262       fprintf(out, " ");
00263       fprintf(out, ctl.qnt_format[iq], atce2[iq] / np);
00264       fprintf(out, " ");
00265       fprintf(out, ctl.qnt_format[iq], rtce2[iq] / np);
00266     }
00267     fprintf(out, " %d\n", np);
00268   }
00269
00270   /* Close file... */
00271   fclose(out);
00272
00273   /* Free... */
00274   free(atm1);
00275   free(atm2);
00276   free(lon1_old);
00277   free(lat1_old);
00278   free(z1_old);
00279   free(lh1);
00280   free(lv1);
00281   free(lon2_old);
00282   free(lat2_old);
00283   free(z2_old);
00284   free(lh2);
00285   free(lv2);
00286
00287   return EXIT_SUCCESS;
00288 }
```

5.9 extract.c File Reference

Extract single trajectory from atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.9.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

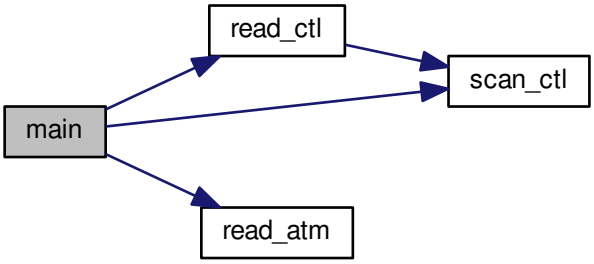
5.9.2 Function Documentation

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

Definition at line 27 of file extract.c.

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

Here is the call graph for this function:



5.10 extract.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   FILE *in, *out;
00036
00037   int f, ip, iq;
00038
00039   /* Allocate... */
00040   ALLOC(atm, atm_t, 1);
00041
00042   /* Check arguments... */
00043   if (argc < 4)
00044     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00045
00046   /* Read control parameters... */
00047   read_ctl(argv[1], argc, argv, &ctl);
00048   ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00049
00050   /* Write info... */
00051   printf("Write trajectory data: %s\n", argv[2]);
00052
00053   /* Create output file... */
00054   if (!(out = fopen(argv[2], "w")))
00055     ERRMSG("Cannot create file!");
00056
00057   /* Write header... */
00058   fprintf(out,
00059           "# $1 = time [s]\n"
```

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

5.11 init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

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

5.11.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

5.11.2 Function Documentation

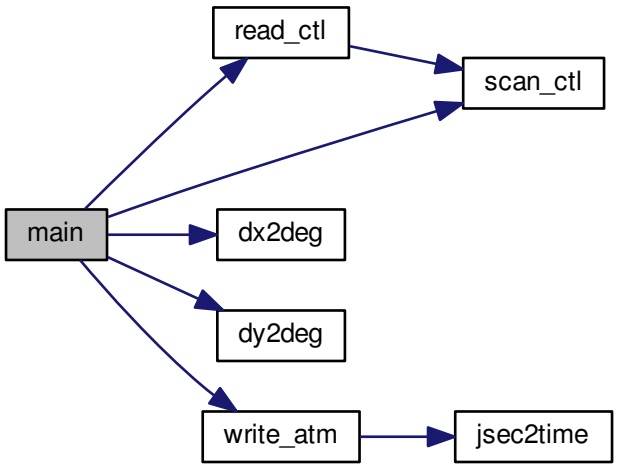
5.11.2.1 int main (int *argc,* char * *argv[ ]*)

Definition at line 27 of file init.c.

```
00029                {
00030
00031   atm_t *atm;
00032
00033   ctl_t ctl;
00034
00035   gsl_rng *rng;
00036
00037   double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038     t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040   int even, ip, irep, rep;
00041
00042   /* Allocate... */
00043   ALLOC(atm, atm_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 3)
00047     ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052   t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053   dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054   z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055   z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056   dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057   lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058   lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059   dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060   lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061   lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062   dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063   st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064   sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065   slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066   slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067   sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068   ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069   uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070   ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071   ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072   even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
00073   rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074   m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075
00076   /* Initialize random number generator... */
00077   gsl_rng_env_setup();
00078   rng = gsl_rng_alloc(gsl_rng_default);
00079
00080   /* Create grid... */
00081   for (t = t0; t <= t1; t += dt)
00082     for (z = z0; z <= z1; z += dz)
00083       for (lon = lon0; lon <= lon1; lon += dlon)
00084         for (lat = lat0; lat <= lat1; lat += dlat)
00085           for (irep = 0; irep < rep; irep++) {
00086
00087             /* Set position... */
00088             atm->time[atm->np]
00089               = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090                  + ut * (gsl_rng_uniform(rng) - 0.5));
00091             atm->p[atm->np]
00092               = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093                   + uz * (gsl_rng_uniform(rng) - 0.5));
00094             atm->lon[atm->np]
00095               = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                  + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00097                  + ulon * (gsl_rng_uniform(rng) - 0.5));
00098             do {
00099               atm->lat[atm->np]
00100                 = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                    + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00102                    + ulat * (gsl_rng_uniform(rng) - 0.5));
00103             } while (even && gsl_rng_uniform(rng) >
00104                      fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106             /* Set particle counter... */
00107             if ((++atm->np) >= NP)
00108               ERRMSG("Too many particles!");
00109           }
00110
00111   /* Check number of air parcels... */
00112   if (atm->np <= 0)
00113     ERRMSG("Did not create any air parcels!");
00114
00115   /* Initialize mass... */
```

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

Here is the call graph for this function:



5.12 init.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   atm_t *atm;
00032
00033   ctl_t ctl;
```

```
00034
00035   gsl_rng *rng;
00036
00037   double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038     t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040   int even, ip, irep, rep;
00041
00042   /* Allocate... */
00043   ALLOC(atm, atm_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 3)
00047     ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052   t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053   dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054   z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055   z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056   dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057   lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058   lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059   dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060   lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061   lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062   dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063   st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064   sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065   slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066   slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067   sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068   ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069   uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070   ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071   ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072   even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
00073   rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074   m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075
00076   /* Initialize random number generator... */
00077   gsl_rng_env_setup();
00078   rng = gsl_rng_alloc(gsl_rng_default);
00079
00080   /* Create grid... */
00081   for (t = t0; t <= t1; t += dt)
00082     for (z = z0; z <= z1; z += dz)
00083       for (lon = lon0; lon <= lon1; lon += dlon)
00084         for (lat = lat0; lat <= lat1; lat += dlat)
00085           for (irep = 0; irep < rep; irep++) {
00086
00087             /* Set position... */
00088             atm->time[atm->np]
00089               = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090                  + ut * (gsl_rng_uniform(rng) - 0.5));
00091             atm->p[atm->np]
00092               = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093                   + uz * (gsl_rng_uniform(rng) - 0.5));
00094             atm->lon[atm->np]
00095               = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                  + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00097                  + ulon * (gsl_rng_uniform(rng) - 0.5));
00098             do {
00099               atm->lat[atm->np]
00100                 = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                    + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00102                    + ulat * (gsl_rng_uniform(rng) - 0.5));
00103             } while (even && gsl_rng_uniform(rng) >
00104                      fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106             /* Set particle counter... */
00107             if ((++atm->np) >= NP)
00108               ERRMSG("Too many particles!");
00109           }
00110
00111   /* Check number of air parcels... */
00112   if (atm->np <= 0)
00113     ERRMSG("Did not create any air parcels!");
00114
00115   /* Initialize mass... */
00116   if (ctl.qnt_m >= 0)
00117     for (ip = 0; ip < atm->np; ip++)
00118       atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120   /* Save data... */
```

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

5.13 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

- int main (int argc, char ∗argv[ ])

5.13.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

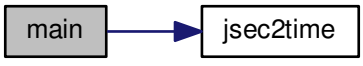
5.13.2 Function Documentation

5.13.2.1 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:



5.14 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 }
```

5.15 libtrac.c File Reference

MPTRAC library definitions.

Functions

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

  Convert Cartesian coordinates to geolocation.
- double clim_hno3 (double t, double lat, double p)

  Climatology of HNO3 volume mixing ratios.
- double clim_tropo (double t, double lat)

  Climatology of tropopause pressure.
- 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 *pt, double *z, 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 *pt, double *z, 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_geopot (ctl_t *ctl, met_t *met)

  Calculate geopotential heights.

- 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.

- void read_met_sample (ctl_t *ctl, met_t *met)

  Downsampling of meteorological data.

- void read_met_tropo (ctl_t *ctl, met_t *met)

  Calculate tropopause pressure.

- 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.

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

Write atmospheric data.

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

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

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

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

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

  Write station data.

5.15.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

5.15.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

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

5.15.2.2 double clim_hno3 (double t, double lat, double p)

Climatology of HNO3 volume mixing ratios.

Definition at line 45 of file libtrac.c.

```
00048            {
00049
00050   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051     9072000.00, 11664000.00, 14342400.00,
00052     16934400.00, 19612800.00, 22291200.00,
00053     24883200.00, 27561600.00, 30153600.00
00054   };
00055
00056   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00057     5, 15, 25, 35, 45, 55, 65, 75, 85
00058   };
00059
00060   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00061     31.6228, 46.4159, 68.1292, 100, 146.78
00062   };
00063
00064   static double hno3[12][18][10] = {
```

```
00065     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00066      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00067      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00068      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00069      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00070      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00071      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00072      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00073      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00074      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00075      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00076      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00077      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00078      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00079      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00080      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00081      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00082      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00083     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00084      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00085      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00086      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00087      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00088      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00089      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00090      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00091      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00092      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00093      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00094      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00095      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00096      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00097      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00098      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00099      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00100      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00101     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00102      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00103      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00104      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00105      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00106      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00107      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00108      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00109      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00110      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00111      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00112      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00113      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00114      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00115      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00116      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00117      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00118      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00119     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00120      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00121      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00122      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00123      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00124      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00125      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00126      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00127      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00128      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00129      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00130      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00131      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00132      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00133      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00134      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00135      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00136      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00137     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00138      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00139      {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00140      {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00141      {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00142      {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00143      {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00144      {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00145      {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00146      {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00147      {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00148      {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00149      {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00150      {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00151      {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
```

```
00152      {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00153      {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00154      {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00155     {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00156      {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00157      {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00158      {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00159      {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00160      {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00161      {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00162      {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00163      {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00164      {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00165      {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00166      {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00167      {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00168      {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00169      {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00170      {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00171      {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00172      {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00173     {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00174      {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00175      {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00176      {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00177      {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00178      {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00179      {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00180      {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00181      {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00182      {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00183      {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00184      {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00185      {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00186      {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00187      {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00188      {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00189      {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00190      {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00191     {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00192      {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00193      {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00194      {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00195      {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00196      {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00197      {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00198      {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00199      {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00200      {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00201      {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00202      {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00203      {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00204      {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00205      {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00206      {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00207      {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00208      {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00209     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00210      {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00211      {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00212      {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00213      {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00214      {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00215      {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00216      {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00217      {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00218      {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00219      {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00220      {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00221      {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00222      {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00223      {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00224      {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00225      {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00226      {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00227     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00228      {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00229      {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00230      {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00231      {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00232      {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00233      {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00234      {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00235      {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00236      {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00237      {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00238      {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
```

```
00239       {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00240       {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00241       {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00242       {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00243       {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00244       {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00245     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00246       {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00247       {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00248       {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00249       {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00250       {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00251       {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00252       {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00253       {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00254       {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00255       {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00256       {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00257       {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00258       {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00259       {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00260       {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00261       {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00262       {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00263     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00264       {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00265       {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00266       {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00267       {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00268       {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00269       {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00270       {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00271       {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00272       {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00273       {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00274       {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00275       {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00276       {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00277       {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00278       {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00279       {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00280       {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00281   };
00282
00283   double aux00, aux01, aux10, aux11, sec;
00284
00285   int ilat, ip, isec;
00286
00287   /* Get seconds since begin of year... */
00288   sec = fmod(t, 365.25 * 86400.);
00289
00290   /* Get indices... */
00291   ilat = locate(lats, 18, lat);
00292   ip = locate(ps, 10, p);
00293   isec = locate(secs, 12, sec);
00294
00295   /* Interpolate... */
00296   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
00297               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
00298   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
00299               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
00300   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
00301               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
00302   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
00303               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
00304   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
00305   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
00306   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
00307 }
```

Here is the call graph for this function:

5.15.2.3 double clim_tropo (double *t*, double *lat*)

Climatology of tropopause pressure.

Definition at line 311 of file libtrac.c.

```
00313             {
00314
00315   static double doys[12]
00316   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
00318   static double lats[73]
00319     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00320     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00321     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00322     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00323     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00324     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00325     75, 77.5, 80, 82.5, 85, 87.5, 90
00326   };
00327
00328   static double tps[12][73]
00329     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00330          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00331          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00332          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00333          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00334          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00335          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00336          275.3, 275.6, 275.4, 274.1, 273.5},
00337   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00338    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00339    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00340    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00341    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00342    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00343    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00344    287.5, 286.2, 285.8},
00345   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00346    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00347    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00348    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00349    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00350    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00351    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00352    304.3, 304.9, 306, 306.6, 306.2, 306},
00353   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00354    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00355    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00356    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00357    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00358    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00359    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00360    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00361   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00362    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00363    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00364    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00365    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00366    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00367    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00368    325.3, 325.8, 325.8},
00369   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00370    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00371    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00372    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00373    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00374    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00375    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00376    308.5, 312.2, 313.1, 313.3},
00377   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00378    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00379    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00380    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00381    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00382    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00383    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00384    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00385   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00386    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00387    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00388    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00389    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00390    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00391    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00392    278.2, 282.6, 287.4, 290.9, 292.5, 293},
00393   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00394    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00395    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00396    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00397    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00398    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00399    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
```

```
00400     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00401    {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00402     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00403     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00404     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00405     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00406     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00407     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00408     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00409     305.1},
00410    {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00411     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00412     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00413     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00414     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00415     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00416     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00417     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00418    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00419     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00420     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00421     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00422     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00423     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00424     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00425     281.7, 281.1, 281.2}
00426    };
00427
00428    double doy, p0, p1, pt;
00429
00430    int imon, ilat;
00431
00432    /* Get day of year... */
00433    doy = fmod(t / 86400., 365.25);
00434    while (doy < 0)
00435      doy += 365.25;
00436
00437    /* Get indices... */
00438    imon = locate(doys, 12, doy);
00439    ilat = locate(lats, 73, lat);
00440
00441    /* Get tropopause pressure... */
00442    p0 = LIN(lats[ilat], tps[imon][ilat],
00443             lats[ilat + 1], tps[imon][ilat + 1], lat);
00444    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
00445             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
00446    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447
00448    /* Return tropopause pressure... */
00449    return pt;
00450  }
```

Here is the call graph for this function:

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

Convert degrees to horizontal distance.

Definition at line 454 of file libtrac.c.

```
00456               {
00457
00458    return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00459  }
```

5.15.2.5 double deg2dy (double *dlat*)

Convert degrees to horizontal distance.

Definition at line 463 of file libtrac.c.

```
00464                {
00465
00466    return dlat * M_PI * RE / 180.;
00467  }
```

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

Convert pressure to vertical distance.

Definition at line 471 of file libtrac.c.

```
00473             {
00474
00475   return -dp * H0 / p;
00476 }
```

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

Convert horizontal distance to degrees.

Definition at line 480 of file libtrac.c.

```
00482               {
00483
00484   /* Avoid singularity at poles... */
00485   if (lat < -89.999 || lat > 89.999)
00486     return 0;
00487   else
00488     return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00489 }
```

5.15.2.8 double dy2deg (double *dy*)

Convert horizontal distance to degrees.

Definition at line 493 of file libtrac.c.

```
00494              {
00495
00496   return dy * 180. / (M_PI * RE);
00497 }
```

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

Convert vertical distance to pressure.

Definition at line 501 of file libtrac.c.

```
00503             {
00504
00505   return -dz * p / H0;
00506 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 510 of file libtrac.c.

```
00514              {
00515
00516   double radius;
00517
00518   radius = z + RE;
00519   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00520   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00521   x[2] = radius * sin(lat / 180 * M_PI);
00522 }
```

5.15.2.11 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 526 of file libtrac.c.

```
00531                 {
00532
00533   char filename[LEN];
00534
00535   static int init;
00536
00537   /* Init... */
00538   if (!init) {
00539     init = 1;
00540
00541     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00542     read_met(ctl, filename, met0);
00543
00544     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00545     read_met(ctl, filename, met1);
00546   }
00547
00548   /* Read new data for forward trajectories... */
00549   if (t > met1->time && ctl->direction == 1) {
00550     memcpy(met0, met1, sizeof(met_t));
00551     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00552     read_met(ctl, filename, met1);
00553   }
00554
00555   /* Read new data for backward trajectories... */
00556   if (t < met0->time && ctl->direction == -1) {
00557     memcpy(met1, met0, sizeof(met_t));
00558     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00559     read_met(ctl, filename, met0);
00560   }
00561 }
```

Here is the call graph for this function:

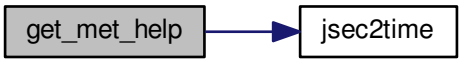
5.15.2.12 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

Get meteorological data for timestep.

Definition at line 565 of file libtrac.c.

```
00570                   {
00571
00572   double t6, r;
00573
00574   int year, mon, day, hour, min, sec;
00575
00576   /* Round time to fixed intervals... */
00577   if (direct == -1)
00578     t6 = floor(t / dt_met) * dt_met;
00579   else
00580     t6 = ceil(t / dt_met) * dt_met;
00581
00582   /* Decode time... */
00583   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00584
00585   /* Set filename... */
00586   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00587 }
```

Here is the call graph for this function:



5.15.2.13 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 591 of file libtrac.c.

```
00597                {
00598
00599   double aux00, aux01, aux10, aux11;
00600
00601   /* Set variables... */
00602   aux00 = array[ix][iy];
00603   aux01 = array[ix][iy + 1];
00604   aux10 = array[ix + 1][iy];
00605   aux11 = array[ix + 1][iy + 1];
00606
00607   /* Interpolate horizontally... */
00608   aux00 = wy * (aux00 - aux01) + aux01;
00609   aux11 = wy * (aux10 - aux11) + aux11;
00610   *var = wx * (aux00 - aux11) + aux11;
00611 }
```

5.15.2.14 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 615 of file libtrac.c.

```
00623                {
00624
00625   double aux00, aux01, aux10, aux11;
00626
00627   /* Interpolate vertically... */
00628   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00629     + array[ix][iy][ip + 1];
00630   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00631     + array[ix][iy + 1][ip + 1];
00632   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00633     + array[ix + 1][iy][ip + 1];
00634   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00635     + array[ix + 1][iy + 1][ip + 1];
00636
00637   /* Interpolate horizontally... */
00638   aux00 = wy * (aux00 - aux01) + aux01;
00639   aux11 = wy * (aux10 - aux11) + aux11;
00640   *var = wx * (aux00 - aux11) + aux11;
00641 }
```

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

Spatial interpolation of meteorological data.

Definition at line 645 of file libtrac.c.

```
00658                {
00659
00660   double wp, wx, wy;
00661
00662   int ip, ix, iy;
00663
00664   /* Check longitude... */
00665   if (met->lon[met->nx - 1] > 180 && lon < 0)
00666     lon += 360;
00667
00668   /* Get indices... */
00669   ip = locate(met->p, met->np, p);
00670   ix = locate(met->lon, met->nx, lon);
00671   iy = locate(met->lat, met->ny, lat);
```

```
00672
00673   /* Get weights... */
00674   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00675   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00676   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00677
00678   /* Interpolate... */
00679   if (ps != NULL)
00680     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00681   if (pt != NULL)
00682     intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00683   if (z != NULL)
00684     intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00685   if (t != NULL)
00686     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00687   if (u != NULL)
00688     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00689   if (v != NULL)
00690     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00691   if (w != NULL)
00692     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00693   if (h2o != NULL)
00694     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00695   if (o3 != NULL)
00696     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00697 }
```

Here is the call graph for this function:

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

Temporal interpolation of meteorological data.

Definition at line 701 of file libtrac.c.

```
00716               {
00717
00718   double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, t0, t1, u0, u1, v0, v1,
00719     w0, w1, wt, z0, z1;
00720
00721   /* Spatial interpolation... */
00722   intpol_met_space(met0, p, lon, lat,
00723                    ps == NULL ? NULL : &ps0,
00724                    pt == NULL ? NULL : &pt0,
00725                    z == NULL ? NULL : &z0,
00726                    t == NULL ? NULL : &t0,
00727                    u == NULL ? NULL : &u0,
00728                    v == NULL ? NULL : &v0,
00729                    w == NULL ? NULL : &w0,
00730                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00731   intpol_met_space(met1, p, lon, lat,
00732                    ps == NULL ? NULL : &ps1,
00733                    pt == NULL ? NULL : &pt1,
00734                    z == NULL ? NULL : &z1,
00735                    t == NULL ? NULL : &t1,
00736                    u == NULL ? NULL : &u1,
00737                    v == NULL ? NULL : &v1,
00738                    w == NULL ? NULL : &w1,
00739                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00740
00741   /* Get weighting factor... */
00742   wt = (met1->time - ts) / (met1->time - met0->time);
00743
00744   /* Interpolate... */
00745   if (ps != NULL)
00746     *ps = wt * (ps0 - ps1) + ps1;
00747   if (pt != NULL)
00748     *pt = wt * (pt0 - pt1) + pt1;
00749   if (z != NULL)
00750     *z = wt * (z0 - z1) + z1;
00751   if (t != NULL)
00752     *t = wt * (t0 - t1) + t1;
00753   if (u != NULL)
00754     *u = wt * (u0 - u1) + u1;
00755   if (v != NULL)
00756     *v = wt * (v0 - v1) + v1;
00757   if (w != NULL)
00758     *w = wt * (w0 - w1) + w1;
```

```
00759   if (h2o != NULL)
00760     *h2o = wt * (h2o0 - h2o1) + h2o1;
00761   if (o3 != NULL)
00762     *o3 = wt * (o30 - o31) + o31;
00763 }
```

Here is the call graph for this function:

5.15.2.17 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 767 of file libtrac.c.

```
00775                   {
00776
00777   struct tm t0, *t1;
00778
00779   time_t jsec0;
00780
00781   t0.tm_year = 100;
00782   t0.tm_mon = 0;
00783   t0.tm_mday = 1;
00784   t0.tm_hour = 0;
00785   t0.tm_min = 0;
00786   t0.tm_sec = 0;
00787
00788   jsec0 = (time_t) jsec + timegm(&t0);
00789   t1 = gmtime(&jsec0);
00790
00791   *year = t1->tm_year + 1900;
00792   *mon = t1->tm_mon + 1;
00793   *day = t1->tm_mday;
00794   *hour = t1->tm_hour;
00795   *min = t1->tm_min;
00796   *sec = t1->tm_sec;
00797   *remain = jsec - floor(jsec);
00798 }
```

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

Find array index.

Definition at line 802 of file libtrac.c.

```
00805             {
00806
00807   int i, ilo, ihi;
00808
00809   ilo = 0;
00810   ihi = n - 1;
00811   i = (ihi + ilo) >> 1;
00812
00813   if (xx[i] < xx[i + 1])
00814     while (ihi > ilo + 1) {
00815       i = (ihi + ilo) >> 1;
00816       if (xx[i] > x)
00817         ihi = i;
00818       else
00819         ilo = i;
00820   } else
00821     while (ihi > ilo + 1) {
00822       i = (ihi + ilo) >> 1;
00823       if (xx[i] <= x)
00824         ihi = i;
00825       else
00826         ilo = i;
00827     }
00828
00829   return ilo;
00830 }
```

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

Read atmospheric data.

Definition at line 834 of file libtrac.c.

```
00837               {
00838
00839   FILE *in;
00840
00841   char line[LEN], *tok;
00842
00843   double t0;
00844
00845   int dimid, ip, iq, ncid, varid;
00846
00847   size_t nparts;
00848
00849   /* Init... */
00850   atm->np = 0;
00851
00852   /* Write info... */
00853   printf("Read atmospheric data: %s\n", filename);
00854
00855   /* Read ASCII data... */
00856   if (ctl->atm_type == 0) {
00857
00858     /* Open file... */
00859     if (!(in = fopen(filename, "r")))
00860       ERRMSG("Cannot open file!");
00861
00862     /* Read line... */
00863     while (fgets(line, LEN, in)) {
00864
00865       /* Read data... */
00866       TOK(line, tok, "%lg", atm->time[atm->np]);
00867       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00868       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00869       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00870       for (iq = 0; iq < ctl->nq; iq++)
00871         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00872
00873       /* Convert altitude to pressure... */
00874       atm->p[atm->np] = P(atm->p[atm->np]);
00875
00876       /* Increment data point counter... */
00877       if ((++atm->np) > NP)
00878         ERRMSG("Too many data points!");
00879     }
00880
00881     /* Close file... */
00882     fclose(in);
00883   }
00884
00885   /* Read binary data... */
00886   else if (ctl->atm_type == 1) {
00887
00888     /* Open file... */
00889     if (!(in = fopen(filename, "r")))
00890       ERRMSG("Cannot open file!");
00891
00892     /* Read data... */
00893     FREAD(&atm->np, int,
00894           1,
00895           in);
00896     FREAD(atm->time, double,
00897             (size_t) atm->np,
00898           in);
00899     FREAD(atm->p, double,
00900             (size_t) atm->np,
00901           in);
00902     FREAD(atm->lon, double,
00903             (size_t) atm->np,
00904           in);
00905     FREAD(atm->lat, double,
00906             (size_t) atm->np,
00907           in);
00908     for (iq = 0; iq < ctl->nq; iq++)
00909       FREAD(atm->q[iq], double,
00910               (size_t) atm->np,
00911             in);
00912
00913     /* Close file... */
```

```
00914     fclose(in);
00915   }
00916
00917   /* Read netCDF data... */
00918   else if (ctl->atm_type == 2) {
00919
00920     /* Open file... */
00921     NC(nc_open(filename, NC_NOWRITE, &ncid));
00922
00923     /* Get dimensions... */
00924     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00925     NC(nc_inq_dimlen(ncid, dimid, &nparts));
00926     atm->np = (int) nparts;
00927     if (atm->np > NP)
00928       ERRMSG("Too many particles!");
00929
00930     /* Get time... */
00931     NC(nc_inq_varid(ncid, "time", &varid));
00932     NC(nc_get_var_double(ncid, varid, &t0));
00933     for (ip = 0; ip < atm->np; ip++)
00934       atm->time[ip] = t0;
00935
00936     /* Read geolocations... */
00937     NC(nc_inq_varid(ncid, "PRESS", &varid));
00938     NC(nc_get_var_double(ncid, varid, atm->p));
00939     NC(nc_inq_varid(ncid, "LON", &varid));
00940     NC(nc_get_var_double(ncid, varid, atm->lon));
00941     NC(nc_inq_varid(ncid, "LAT", &varid));
00942     NC(nc_get_var_double(ncid, varid, atm->lat));
00943
00944     /* Read variables... */
00945     if (ctl->qnt_p >= 0)
00946       if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
00947         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
00948     if (ctl->qnt_t >= 0)
00949       if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
00950         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
00951     if (ctl->qnt_u >= 0)
00952       if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
00953         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
00954     if (ctl->qnt_v >= 0)
00955       if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
00956         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
00957     if (ctl->qnt_w >= 0)
00958       if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
00959         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
00960     if (ctl->qnt_h2o >= 0)
00961       if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
00962         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
00963     if (ctl->qnt_o3 >= 0)
00964       if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
00965         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
00966     if (ctl->qnt_theta >= 0)
00967       if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
00968         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
00969     if (ctl->qnt_pv >= 0)
00970       if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
00971         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
00972
00973     /* Check data... */
00974     for (ip = 0; ip < atm->np; ip++)
00975       if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
00976           || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
00977           || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
00978           || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
00979           || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
00980         atm->time[ip] = GSL_NAN;
00981         atm->p[ip] = GSL_NAN;
00982         atm->lon[ip] = GSL_NAN;
00983         atm->lat[ip] = GSL_NAN;
00984         for (iq = 0; iq < ctl->nq; iq++)
00985           atm->q[iq][ip] = GSL_NAN;
00986       } else {
00987         if (ctl->qnt_h2o >= 0)
00988           atm->q[ctl->qnt_h2o][ip] *= 1.608;
00989         if (atm->lon[ip] > 180)
00990           atm->lon[ip] -= 360;
00991       }
00992
00993     /* Close file... */
00994     NC(nc_close(ncid));
00995   }
00996
00997   /* Error... */
00998   else
00999     ERRMSG("Atmospheric data type not supported!");
01000
```

```
01001   /* Check number of points... */
01002   if (atm->np < 1)
01003     ERRMSG("Can not read any data!");
01004 }
```

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

Read control parameters.

Definition at line 1008 of file libtrac.c.

```
01012               {
01013
01014   int ip, iq;
01015
01016   /* Write info... */
01017   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01018          "(executable: %s | compiled: %s, %s)\n\n",
01019          argv[0], __DATE__, __TIME__);
01020
01021   /* Initialize quantity indices... */
01022   ctl->qnt_ens = -1;
01023   ctl->qnt_m = -1;
01024   ctl->qnt_r = -1;
01025   ctl->qnt_rho = -1;
01026   ctl->qnt_ps = -1;
01027   ctl->qnt_pt = -1;
01028   ctl->qnt_z = -1;
01029   ctl->qnt_p = -1;
01030   ctl->qnt_t = -1;
01031   ctl->qnt_u = -1;
01032   ctl->qnt_v = -1;
01033   ctl->qnt_w = -1;
01034   ctl->qnt_h2o = -1;
01035   ctl->qnt_o3 = -1;
01036   ctl->qnt_theta = -1;
01037   ctl->qnt_pv = -1;
01038   ctl->qnt_tice = -1;
01039   ctl->qnt_tsts = -1;
01040   ctl->qnt_tnat = -1;
01041   ctl->qnt_gw_var = -1;
01042   ctl->qnt_stat = -1;
01043
01044   /* Read quantities... */
01045   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01046   if (ctl->nq > NQ)
01047     ERRMSG("Too many quantities!");
01048   for (iq = 0; iq < ctl->nq; iq++) {
01049
01050     /* Read quantity name and format... */
01051     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01052     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01053              ctl->qnt_format[iq]);
01054
01055     /* Try to identify quantity... */
01056     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01057       ctl->qnt_ens = iq;
01058       sprintf(ctl->qnt_unit[iq], "-");
01059     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01060       ctl->qnt_m = iq;
01061       sprintf(ctl->qnt_unit[iq], "kg");
01062     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01063       ctl->qnt_r = iq;
01064       sprintf(ctl->qnt_unit[iq], "m");
01065     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01066       ctl->qnt_rho = iq;
01067       sprintf(ctl->qnt_unit[iq], "kg/m^3");
01068     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01069       ctl->qnt_ps = iq;
01070       sprintf(ctl->qnt_unit[iq], "hPa");
01071     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01072       ctl->qnt_pt = iq;
01073       sprintf(ctl->qnt_unit[iq], "hPa");
01074     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01075       ctl->qnt_z = iq;
01076       sprintf(ctl->qnt_unit[iq], "km");
01077     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01078       ctl->qnt_p = iq;
01079       sprintf(ctl->qnt_unit[iq], "hPa");
01080     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
```

```
01081       ctl->qnt_t = iq;
01082       sprintf(ctl->qnt_unit[iq], "K");
01083     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01084       ctl->qnt_u = iq;
01085       sprintf(ctl->qnt_unit[iq], "m/s");
01086     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01087       ctl->qnt_v = iq;
01088       sprintf(ctl->qnt_unit[iq], "m/s");
01089     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01090       ctl->qnt_w = iq;
01091       sprintf(ctl->qnt_unit[iq], "hPa/s");
01092     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01093       ctl->qnt_h2o = iq;
01094       sprintf(ctl->qnt_unit[iq], "1");
01095     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01096       ctl->qnt_o3 = iq;
01097       sprintf(ctl->qnt_unit[iq], "1");
01098     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01099       ctl->qnt_theta = iq;
01100       sprintf(ctl->qnt_unit[iq], "K");
01101     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01102       ctl->qnt_pv = iq;
01103       sprintf(ctl->qnt_unit[iq], "PVU");
01104     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01105       ctl->qnt_tice = iq;
01106       sprintf(ctl->qnt_unit[iq], "K");
01107     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01108       ctl->qnt_tsts = iq;
01109       sprintf(ctl->qnt_unit[iq], "K");
01110     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01111       ctl->qnt_tnat = iq;
01112       sprintf(ctl->qnt_unit[iq], "K");
01113     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
01114       ctl->qnt_gw_var = iq;
01115       sprintf(ctl->qnt_unit[iq], "K^2");
01116     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01117       ctl->qnt_stat = iq;
01118       sprintf(ctl->qnt_unit[iq], "-");
01119     } else
01120       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01121   }
01122
01123   /* Time steps of simulation... */
01124   ctl->direction =
01125     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01126   if (ctl->direction != -1 && ctl->direction != 1)
01127     ERRMSG("Set DIRECTION to -1 or 1!");
01128   ctl->t_start =
01129     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
01130   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
01131   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01132
01133   /* Meteorological data... */
01134   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01135   ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01136   ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01137   ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01138   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01139   if (ctl->met_np > EP)
01140     ERRMSG("Too many levels!");
01141   for (ip = 0; ip < ctl->met_np; ip++)
01142     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01143   ctl->met_tropo
01144     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01145   scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01146   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01147
01148   /* Isosurface parameters... */
01149   ctl->isosurf
01150     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01151   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01152
01153   /* Diffusion parameters... */
01154   ctl->turb_dx_trop
01155     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
01156   ctl->turb_dx_strat
01157     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
01158   ctl->turb_dz_trop
01159     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
01160   ctl->turb_dz_strat
01161     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01162   ctl->turb_meso =
01163     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
01164
01165   /* Life time of particles... */
01166   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01167   ctl->tdec_strat =
```

```
01168     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01169
01170   /* PSC analysis... */
01171   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01172   ctl->psc_hno3 =
01173     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01174
01175   /* Gravity wave analysis... */
01176   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
01177
01178   /* Output of atmospheric data... */
01179   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
01180   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01181   ctl->atm_dt_out =
01182     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01183   ctl->atm_filter =
01184     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01185   ctl->atm_type =
01186     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01187
01188   /* Output of CSI data... */
01189   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01190   ctl->csi_dt_out =
01191     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01192   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
01193            ctl->csi_obsfile);
01194   ctl->csi_obsmin =
01195     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01196   ctl->csi_modmin =
01197     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01198   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01199   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01200   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01201   ctl->csi_lon0 =
01202     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01203   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01204   ctl->csi_nx =
01205     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01206   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01207   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01208   ctl->csi_ny =
01209     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01210
01211   /* Output of ensemble data... */
01212   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01213
01214   /* Output of grid data... */
01215   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01216            ctl->grid_basename);
01217   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01218   ctl->grid_dt_out =
01219     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01220   ctl->grid_sparse =
01221     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01222   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01223   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01224   ctl->grid_nz =
01225     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01226   ctl->grid_lon0 =
01227     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01228   ctl->grid_lon1 =
01229     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01230   ctl->grid_nx =
01231     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01232   ctl->grid_lat0 =
01233     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01234   ctl->grid_lat1 =
01235     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01236   ctl->grid_ny =
01237     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01238
01239   /* Output of profile data... */
01240   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01241            ctl->prof_basename);
01242   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
01243   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01244   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01245   ctl->prof_nz =
01246     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01247   ctl->prof_lon0 =
01248     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
```

```
01249   ctl->prof_lon1 =
01250     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01251   ctl->prof_nx =
01252     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01253   ctl->prof_lat0 =
01254     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01255   ctl->prof_lat1 =
01256     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01257   ctl->prof_ny =
01258     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01259
01260   /* Output of station data... */
01261   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01262            ctl->stat_basename);
01263   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01264   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01265   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01266 }
```

Here is the call graph for this function:

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

Read meteorological data file.

Definition at line 1270 of file libtrac.c.

```
01273               {
01274
01275   char cmd[2 * LEN], levname[LEN], tstr[10];
01276
01277   static float help[EX * EY];
01278
01279   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01280
01281   size_t np, nx, ny;
01282
01283   /* Write info... */
01284   printf("Read meteorological data: %s\n", filename);
01285
01286   /* Get time from filename... */
01287   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01288   year = atoi(tstr);
01289   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01290   mon = atoi(tstr);
01291   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01292   day = atoi(tstr);
01293   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01294   hour = atoi(tstr);
01295   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01296
01297   /* Open netCDF file... */
01298   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01299
01300     /* Try to stage meteo file... */
01301     STOP_TIMER(TIMER_INPUT);
01302     START_TIMER(TIMER_STAGE);
01303     if (ctl->met_stage[0] != '-') {
01304       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01305               year, mon, day, hour, filename);
01306       if (system(cmd) != 0)
01307         ERRMSG("Error while staging meteo data!");
```

```
01308     }
01309     STOP_TIMER(TIMER_STAGE);
01310     START_TIMER(TIMER_INPUT);
01311
01312     /* Try to open again... */
01313     NC(nc_open(filename, NC_NOWRITE, &ncid));
01314   }
01315
01316   /* Get dimensions... */
01317   NC(nc_inq_dimid(ncid, "lon", &dimid));
01318   NC(nc_inq_dimlen(ncid, dimid, &nx));
01319   if (nx < 2 || nx > EX)
01320     ERRMSG("Number of longitudes out of range!");
01321
01322   NC(nc_inq_dimid(ncid, "lat", &dimid));
01323   NC(nc_inq_dimlen(ncid, dimid, &ny));
01324   if (ny < 2 || ny > EY)
01325     ERRMSG("Number of latitudes out of range!");
01326
01327   sprintf(levname, "lev");
01328   NC(nc_inq_dimid(ncid, levname, &dimid));
01329   NC(nc_inq_dimlen(ncid, dimid, &np));
01330   if (np == 1) {
01331     sprintf(levname, "lev_2");
01332     NC(nc_inq_dimid(ncid, levname, &dimid));
01333     NC(nc_inq_dimlen(ncid, dimid, &np));
01334   }
01335   if (np < 2 || np > EP)
01336     ERRMSG("Number of levels out of range!");
01337
01338   /* Store dimensions... */
01339   met->np = (int) np;
01340   met->nx = (int) nx;
01341   met->ny = (int) ny;
01342
01343   /* Get horizontal grid... */
01344   NC(nc_inq_varid(ncid, "lon", &varid));
01345   NC(nc_get_var_double(ncid, varid, met->lon));
01346   NC(nc_inq_varid(ncid, "lat", &varid));
01347   NC(nc_get_var_double(ncid, varid, met->lat));
01348
01349   /* Read meteorological data... */
01350   read_met_help(ncid, "t", "T", met, met->t, 1.0);
01351   read_met_help(ncid, "u", "U", met, met->u, 1.0);
01352   read_met_help(ncid, "v", "V", met, met->v, 1.0);
01353   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01354   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
01355   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
01356
01357   /* Meteo data on pressure levels... */
01358   if (ctl->met_np <= 0) {
01359
01360     /* Read pressure levels from file... */
01361     NC(nc_inq_varid(ncid, levname, &varid));
01362     NC(nc_get_var_double(ncid, varid, met->p));
01363     for (ip = 0; ip < met->np; ip++)
01364       met->p[ip] /= 100.;
01365
01366     /* Extrapolate data for lower boundary... */
01367     read_met_extrapolate(met);
01368   }
01369
01370   /* Meteo data on model levels... */
01371   else {
01372
01373     /* Read pressure data from file... */
01374     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01375
01376     /* Interpolate from model levels to pressure levels... */
01377     read_met_ml2pl(ctl, met, met->t);
01378     read_met_ml2pl(ctl, met, met->u);
01379     read_met_ml2pl(ctl, met, met->v);
01380     read_met_ml2pl(ctl, met, met->w);
01381     read_met_ml2pl(ctl, met, met->h2o);
01382     read_met_ml2pl(ctl, met, met->o3);
01383
01384     /* Set pressure levels... */
01385     met->np = ctl->met_np;
01386     for (ip = 0; ip < met->np; ip++)
01387       met->p[ip] = ctl->met_p[ip];
01388   }
01389
01390   /* Check ordering of pressure levels... */
01391   for (ip = 1; ip < met->np; ip++)
01392     if (met->p[ip - 1] < met->p[ip])
01393       ERRMSG("Pressure levels must be descending!");
01394
```

```
01395   /* Read surface pressure... */
01396   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01397       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01398     NC(nc_get_var_float(ncid, varid, help));
01399     for (iy = 0; iy < met->ny; iy++)
01400       for (ix = 0; ix < met->nx; ix++)
01401         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01402   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01403              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01404     NC(nc_get_var_float(ncid, varid, help));
01405     for (iy = 0; iy < met->ny; iy++)
01406       for (ix = 0; ix < met->nx; ix++)
01407         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01408   } else
01409     for (ix = 0; ix < met->nx; ix++)
01410       for (iy = 0; iy < met->ny; iy++)
01411         met->ps[ix][iy] = met->p[0];
01412
01413   /* Create periodic boundary conditions... */
01414   read_met_periodic(met);
01415
01416   /* Calculate geopotential heights... */
01417   read_met_geopot(ctl, met);
01418
01419   /* Calculate tropopause pressure... */
01420   read_met_tropo(ctl, met);
01421
01422   /* Downsampling... */
01423   read_met_sample(ctl, met);
01424
01425   /* Close file... */
01426   NC(nc_close(ncid));
01427 }
```

Here is the call graph for this function:

5.15.2.22 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 1431 of file libtrac.c.

```
01432               {
01433
01434   int ip, ip0, ix, iy;
01435
01436   /* Loop over columns... */
01437   for (ix = 0; ix < met->nx; ix++)
01438     for (iy = 0; iy < met->ny; iy++) {
01439
01440       /* Find lowest valid data point... */
01441       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01442         if (!gsl_finite(met->t[ix][iy][ip0])
01443             || !gsl_finite(met->u[ix][iy][ip0])
01444             || !gsl_finite(met->v[ix][iy][ip0])
01445             || !gsl_finite(met->w[ix][iy][ip0]))
01446           break;
01447
01448       /* Extrapolate... */
01449       for (ip = ip0; ip >= 0; ip--) {
01450         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01451         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01452         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01453         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01454         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01455         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01456       }
01457     }
01458 }
```

5.15.2.23 void read_met_geopot (ctl_t * *ctl*, met_t * *met*)

Calculate geopotential heights.

Definition at line 1462 of file libtrac.c.

```
01464               {
01465
01466   static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01467
01468   static int init, topo_nx = -1, topo_ny;
01469
01470   FILE *in;
01471
01472   char line[LEN];
01473
01474   double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01475
01476   float help[EX][EY];
01477
01478   int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01479
01480   /* Initialize geopotential heights... */
01481   for (ix = 0; ix < met->nx; ix++)
01482     for (iy = 0; iy < met->ny; iy++)
01483       for (ip = 0; ip < met->np; ip++)
01484         met->z[ix][iy][ip] = GSL_NAN;
01485
01486   /* Check filename... */
01487   if (ctl->met_geopot[0] == '-')
01488     return;
01489
01490   /* Read surface geopotential... */
01491   if (!init) {
01492
01493     /* Write info... */
01494     printf("Read surface geopotential: %s\n", ctl->met_geopot);
01495
01496     /* Open file... */
01497     if (!(in = fopen(ctl->met_geopot, "r")))
01498       ERRMSG("Cannot open file!");
01499
01500     /* Read data... */
01501     while (fgets(line, LEN, in))
01502       if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01503         if (rlon != rlon_old) {
01504           if ((++topo_nx) >= EX)
01505             ERRMSG("Too many longitudes!");
01506           topo_ny = 0;
01507         }
01508         rlon_old = rlon;
01509         topo_lon[topo_nx] = rlon;
01510         topo_lat[topo_ny] = rlat;
01511         topo_z[topo_nx][topo_ny] = rz;
01512         if ((++topo_ny) >= EY)
01513           ERRMSG("Too many latitudes!");
01514       }
01515     if ((++topo_nx) >= EX)
01516       ERRMSG("Too many longitudes!");
01517
01518     /* Close file... */
01519     fclose(in);
01520
01521     /* Check grid spacing... */
01522     if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01523         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01524       ERRMSG("Grid spacing does not match!");
01525
01526     /* Set init flag... */
01527     init = 1;
01528   }
01529
01530   /* Apply hydrostatic equation to calculate geopotential heights... */
01531   for (ix = 0; ix < met->nx; ix++)
01532     for (iy = 0; iy < met->ny; iy++) {
01533
01534       /* Get surface height... */
01535       lon = met->lon[ix];
01536       if (lon < topo_lon[0])
01537         lon += 360;
01538       else if (lon > topo_lon[topo_nx - 1])
01539         lon -= 360;
01540       lat = met->lat[iy];
```

```
01541       tx = locate(topo_lon, topo_nx, lon);
01542       ty = locate(topo_lat, topo_ny, lat);
01543       z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01544                topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01545       z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01546                topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01547       z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01548
01549       /* Find surface pressure level... */
01550       ip0 = locate(met->p, met->np, met->ps[ix][iy]);
01551
01552       /* Get surface temperature... */
01553       ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
01554                met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
01555
01556       /* Upper part of profile... */
01557       met->z[ix][iy][ip0 + 1]
01558         = (float) (z0 + 8.31441 / 28.9647 / G0
01559                    * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01560                    * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01561       for (ip = ip0 + 2; ip < met->np; ip++)
01562         met->z[ix][iy][ip]
01563           = (float) (met->z[ix][iy][ip - 1] + 8.31441 / 28.9647 / G0
01564                      * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01565                      * log(met->p[ip - 1] / met->p[ip]));
01566     }
01567
01568   /* Smooth fields... */
01569   for (ip = 0; ip < met->np; ip++) {
01570
01571     /* Median filter... */
01572     for (ix = 0; ix < met->nx; ix++)
01573       for (iy = 0; iy < met->nx; iy++) {
01574         n = 0;
01575         for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
01576           ix3 = ix2;
01577           if (ix3 < 0)
01578             ix3 += met->nx;
01579           if (ix3 >= met->nx)
01580             ix3 -= met->nx;
01581           for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01582                iy2++)
01583             if (gsl_finite(met->z[ix3][iy2][ip])) {
01584               data[n] = met->z[ix3][iy2][ip];
01585               n++;
01586             }
01587         }
01588         if (n > 0) {
01589           gsl_sort(data, 1, (size_t) n);
01590           help[ix][iy] = (float)
01591             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01592         } else
01593           help[ix][iy] = GSL_NAN;
01594       }
01595
01596     /* Copy data... */
01597     for (ix = 0; ix < met->nx; ix++)
01598       for (iy = 0; iy < met->nx; iy++)
01599         met->z[ix][iy][ip] = help[ix][iy];
01600   }
01601 }
```

Here is the call graph for this function:

5.15.2.24 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 1605 of file libtrac.c.

```
01611             {
01612
01613   static float help[EX * EY * EP];
01614
01615   int ip, ix, iy, n = 0, varid;
01616
01617   /* Check if variable exists... */
01618   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01619     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
```

```
01620       return;
01621
01622   /* Read data... */
01623   NC(nc_get_var_float(ncid, varid, help));
01624
01625   /* Copy and check data... */
01626   for (ip = 0; ip < met->np; ip++)
01627     for (iy = 0; iy < met->ny; iy++)
01628       for (ix = 0; ix < met->nx; ix++) {
01629         dest[ix][iy][ip] = scl * help[n++];
01630         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
01631           dest[ix][iy][ip] = GSL_NAN;
01632       }
01633 }
```

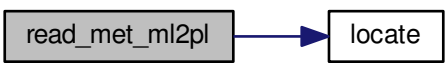
5.15.2.25 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 1637 of file libtrac.c.

```
01640                           {
01641
01642   double aux[EP], p[EP], pt;
01643
01644   int ip, ip2, ix, iy;
01645
01646   /* Loop over columns... */
01647   for (ix = 0; ix < met->nx; ix++)
01648     for (iy = 0; iy < met->ny; iy++) {
01649
01650       /* Copy pressure profile... */
01651       for (ip = 0; ip < met->np; ip++)
01652         p[ip] = met->pl[ix][iy][ip];
01653
01654       /* Interpolate... */
01655       for (ip = 0; ip < ctl->met_np; ip++) {
01656         pt = ctl->met_p[ip];
01657         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01658           pt = p[0];
01659         else if ((pt > p[met->np - 1] && p[1] > p[0])
01660                  || (pt < p[met->np - 1] && p[1] < p[0]))
01661           pt = p[met->np - 1];
01662         ip2 = locate(p, met->np, pt);
01663         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01664                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01665       }
01666
01667       /* Copy data... */
01668       for (ip = 0; ip < ctl->met_np; ip++)
01669         var[ix][iy][ip] = (float) aux[ip];
01670     }
01671 }
```

Here is the call graph for this function:



5.15.2.26 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 1675 of file libtrac.c.

```
01676               {
01677
01678   int ip, iy;
01679
01680   /* Check longitudes... */
01681   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01682              + met->lon[1] - met->lon[0] - 360) < 0.01))
01683     return;
01684
01685   /* Increase longitude counter... */
01686   if ((++met->nx) > EX)
01687     ERRMSG("Cannot create periodic boundary conditions!");
01688
01689   /* Set longitude... */
01690   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01691
01692   /* Loop over latitudes and pressure levels... */
01693   for (iy = 0; iy < met->ny; iy++)
01694     for (ip = 0; ip < met->np; ip++) {
01695       met->ps[met->nx - 1][iy] = met->ps[0][iy];
01696       met->pt[met->nx - 1][iy] = met->pt[0][iy];
01697       met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01698       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01699       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01700       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01701       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01702       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01703       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01704     }
01705 }
```

5.15.2.27 void read_met_sample (ctl_t * *ctl*, met_t * *met*)

Downsampling of meteorological data.

Definition at line 1709 of file libtrac.c.

```
01711               {
01712
01713   met_t *help;
01714
01715   float w, wsum;
01716
01717   int ip, ip2, ix, ix2, iy, iy2;
01718
01719   /* Check parameters... */
01720   if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1)
01721     return;
01722
01723   /* Allocate... */
01724   ALLOC(help, met_t, 1);
01725
01726   /* Copy data... */
01727   help->nx = met->nx;
01728   help->ny = met->ny;
01729   help->np = met->np;
01730   memcpy(help->lon, met->lon, sizeof(met->lon));
01731   memcpy(help->lat, met->lat, sizeof(met->lat));
01732   memcpy(help->p, met->p, sizeof(met->p));
01733
01734   /* Smoothing... */
01735   for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01736     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01737       for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01738         help->ps[ix][iy] = 0;
01739         help->pt[ix][iy] = 0;
01740         help->z[ix][iy][ip] = 0;
01741         help->t[ix][iy][ip] = 0;
01742         help->u[ix][iy][ip] = 0;
```

```
01743         help->v[ix][iy][ip] = 0;
01744         help->w[ix][iy][ip] = 0;
01745         help->h2o[ix][iy][ip] = 0;
01746         help->o3[ix][iy][ip] = 0;
01747         wsum = 0;
01748         for (ix2 = GSL_MAX(ix - ctl->met_dx + 1, 0);
01749              ix2 <= GSL_MIN(ix + ctl->met_dx - 1, met->nx - 1); ix2++)
01750           for (iy2 = GSL_MAX(iy - ctl->met_dy + 1, 0);
01751                iy2 <= GSL_MIN(iy + ctl->met_dy - 1, met->ny - 1); iy2++)
01752             for (ip2 = GSL_MAX(ip - ctl->met_dp + 1, 0);
01753                  ip2 <= GSL_MIN(ip + ctl->met_dp - 1, met->np - 1); ip2++) {
01754               w = (float) (1.0 - fabs(ix - ix2) / ctl->met_dx)
01755                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_dy)
01756                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_dp);
01757               help->ps[ix][iy] += w * met->ps[ix2][iy2];
01758               help->pt[ix][iy] += w * met->pt[ix2][iy2];
01759               help->z[ix][iy][ip] += w * met->z[ix2][iy2][ip2];
01760               help->t[ix][iy][ip] += w * met->t[ix2][iy2][ip2];
01761               help->u[ix][iy][ip] += w * met->u[ix2][iy2][ip2];
01762               help->v[ix][iy][ip] += w * met->v[ix2][iy2][ip2];
01763               help->w[ix][iy][ip] += w * met->w[ix2][iy2][ip2];
01764               help->h2o[ix][iy][ip] += w * met->h2o[ix2][iy2][ip2];
01765               help->o3[ix][iy][ip] += w * met->o3[ix2][iy2][ip2];
01766               wsum += w;
01767             }
01768         help->ps[ix][iy] /= wsum;
01769         help->pt[ix][iy] /= wsum;
01770         help->t[ix][iy][ip] /= wsum;
01771         help->z[ix][iy][ip] /= wsum;
01772         help->u[ix][iy][ip] /= wsum;
01773         help->v[ix][iy][ip] /= wsum;
01774         help->w[ix][iy][ip] /= wsum;
01775         help->h2o[ix][iy][ip] /= wsum;
01776         help->o3[ix][iy][ip] /= wsum;
01777       }
01778     }
01779   }
01780
01781   /* Downsampling... */
01782   met->nx = 0;
01783   for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01784     met->lon[met->nx] = help->lon[ix];
01785     met->ny = 0;
01786     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01787       met->lat[met->ny] = help->lat[iy];
01788       met->ps[met->nx][met->ny] = help->ps[ix][iy];
01789       met->pt[met->nx][met->ny] = help->pt[ix][iy];
01790       met->np = 0;
01791       for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01792         met->p[met->np] = help->p[ip];
01793         met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01794         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01795         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01796         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01797         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01798         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01799         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01800         met->np++;
01801       }
01802       met->ny++;
01803     }
01804     met->nx++;
01805   }
01806
01807   /* Free... */
01808   free(help);
01809 }
```

5.15.2.28 void read_met_tropo (ctl_t * *ctl*, met_t * *met*)

Calculate tropopause pressure.

Definition at line 1813 of file libtrac.c.

```
01815               {
01816
01817   gsl_interp_accel *acc;
01818
01819   gsl_spline *spline;
01820
```

```
01821   double tt[400], tt2[400], tz[400], tz2[400];
01822
01823   int found, ix, iy, iz, iz2;
01824
01825   /* Allocate... */
01826   acc = gsl_interp_accel_alloc();
01827   spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01828
01829   /* Do not calculate tropopause... */
01830   if (ctl->met_tropo == 0)
01831     for (ix = 0; ix < met->nx; ix++)
01832       for (iy = 0; iy < met->ny; iy++)
01833         met->pt[ix][iy] = GSL_NAN;
01834
01835   /* Use tropopause climatology... */
01836   else if (ctl->met_tropo == 1)
01837     for (ix = 0; ix < met->nx; ix++)
01838       for (iy = 0; iy < met->ny; iy++)
01839         met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01840
01841   /* Use cold point... */
01842   else if (ctl->met_tropo == 2) {
01843
01844     /* Loop over grid points... */
01845     for (ix = 0; ix < met->nx; ix++)
01846       for (iy = 0; iy < met->ny; iy++) {
01847
01848         /* Get temperature profile... */
01849         for (iz = 0; iz < met->np; iz++) {
01850           tz[iz] = Z(met->p[iz]);
01851           tt[iz] = met->t[ix][iy][iz];
01852         }
01853
01854         /* Interpolate temperature profile... */
01855         gsl_spline_init(spline, tz, tt, (size_t) met->np);
01856         for (iz = 0; iz <= 170; iz++) {
01857           tz2[iz] = 4.5 + 0.1 * iz;
01858           tt2[iz] = gsl_spline_eval(spline, tz2[iz], acc);
01859         }
01860
01861         /* Find minimum... */
01862         iz = (int) gsl_stats_min_index(tt2, 1, 171);
01863         if (iz <= 0 || iz >= 170)
01864           met->pt[ix][iy] = GSL_NAN;
01865         else
01866           met->pt[ix][iy] = P(tz2[iz]);
01867       }
01868   }
01869
01870   /* Use WMO definition... */
01871   else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
01872
01873     /* Loop over grid points... */
01874     for (ix = 0; ix < met->nx; ix++)
01875       for (iy = 0; iy < met->ny; iy++) {
01876
01877         /* Get temperature profile... */
01878         for (iz = 0; iz < met->np; iz++) {
01879           tz[iz] = Z(met->p[iz]);
01880           tt[iz] = met->t[ix][iy][iz];
01881         }
01882
01883         /* Interpolate temperature profile... */
01884         gsl_spline_init(spline, tz, tt, (size_t) met->np);
01885         for (iz = 0; iz <= 160; iz++) {
01886           tz2[iz] = 4.5 + 0.1 * iz;
01887           tt2[iz] = gsl_spline_eval(spline, tz2[iz], acc);
01888         }
01889
01890         /* Find 1st tropopause... */
01891         met->pt[ix][iy] = GSL_NAN;
01892         for (iz = 0; iz <= 140; iz++) {
01893           found = 1;
01894           for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
01895             if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01896                 / log(P(tz2[iz2]) / P(tz2[iz])) > 2.0) {
01897               found = 0;
01898               break;
01899             }
01900           if (found) {
01901             if (iz > 0 && iz < 140)
01902               met->pt[ix][iy] = P(tz2[iz]);
01903             break;
01904           }
01905         }
01906
01907         /* Find 2nd tropopause... */
```

```
01908         if (ctl->met_tropo == 4) {
01909           met->pt[ix][iy] = GSL_NAN;
01910           for (; iz <= 140; iz++) {
01911             found = 1;
01912             for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
01913               if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01914                   / log(P(tz2[iz2]) / P(tz2[iz])) < 3.0) {
01915                 found = 0;
01916                 break;
01917               }
01918             if (found)
01919               break;
01920           }
01921           for (; iz <= 140; iz++) {
01922             found = 1;
01923             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
01924               if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01925                   / log(P(tz2[iz2]) / P(tz2[iz])) > 2.0) {
01926                 found = 0;
01927                 break;
01928               }
01929             if (found) {
01930               if (iz > 0 && iz < 140)
01931                 met->pt[ix][iy] = P(tz2[iz]);
01932               break;
01933             }
01934           }
01935         }
01936       }
01937   }
01938
01939   else
01940     ERRMSG("Cannot calculate tropopause!");
01941
01942   /* Free... */
01943   gsl_spline_free(spline);
01944   gsl_interp_accel_free(acc);
01945 }
```

Here is the call graph for this function:

5.15.2.29 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 1949 of file libtrac.c.

```
01956               {
01957
01958   FILE *in = NULL;
01959
01960   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01961     msg[LEN], rvarname[LEN], rval[LEN];
01962
01963   int contain = 0, i;
01964
01965   /* Open file... */
01966   if (filename[strlen(filename) - 1] != '-')
01967     if (!(in = fopen(filename, "r")))
01968       ERRMSG("Cannot open file!");
01969
01970   /* Set full variable name... */
01971   if (arridx >= 0) {
01972     sprintf(fullname1, "%s[%d]", varname, arridx);
01973     sprintf(fullname2, "%s[*]", varname);
01974   } else {
01975     sprintf(fullname1, "%s", varname);
01976     sprintf(fullname2, "%s", varname);
01977   }
01978
01979   /* Read data... */
01980   if (in != NULL)
01981     while (fgets(line, LEN, in))
01982       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01983         if (strcasecmp(rvarname, fullname1) == 0 ||
01984             strcasecmp(rvarname, fullname2) == 0) {
01985           contain = 1;
01986           break;
```

```
01987         }
01988     for (i = 1; i < argc - 1; i++)
01989       if (strcasecmp(argv[i], fullname1) == 0 ||
01990           strcasecmp(argv[i], fullname2) == 0) {
01991         sprintf(rval, "%s", argv[i + 1]);
01992         contain = 1;
01993         break;
01994       }
01995
01996     /* Close file... */
01997     if (in != NULL)
01998       fclose(in);
01999
02000     /* Check for missing variables... */
02001     if (!contain) {
02002       if (strlen(defvalue) > 0)
02003         sprintf(rval, "%s", defvalue);
02004       else {
02005         sprintf(msg, "Missing variable %s!\n", fullname1);
02006         ERRMSG(msg);
02007       }
02008     }
02009
02010     /* Write info... */
02011     printf("%s = %s\n", fullname1, rval);
02012
02013     /* Return values... */
02014     if (value != NULL)
02015       sprintf(value, "%s", rval);
02016     return atof(rval);
02017 }
```

5.15.2.30 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 2021 of file libtrac.c.

```
02029                 {
02030
02031     struct tm t0, t1;
02032
02033     t0.tm_year = 100;
02034     t0.tm_mon = 0;
02035     t0.tm_mday = 1;
02036     t0.tm_hour = 0;
02037     t0.tm_min = 0;
02038     t0.tm_sec = 0;
02039
02040     t1.tm_year = year - 1900;
02041     t1.tm_mon = mon - 1;
02042     t1.tm_mday = day;
02043     t1.tm_hour = hour;
02044     t1.tm_min = min;
02045     t1.tm_sec = sec;
02046
02047     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02048 }
```

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

Measure wall-clock time.

Definition at line 2052 of file libtrac.c.

```
02055             {
02056
02057     static double starttime[NTIMER], runtime[NTIMER];
02058
02059     /* Check id... */
02060     if (id < 0 || id >= NTIMER)
02061       ERRMSG("Too many timers!");
02062
02063     /* Start timer... */
```

```
02064   if (mode == 1) {
02065     if (starttime[id] <= 0)
02066       starttime[id] = omp_get_wtime();
02067     else
02068       ERRMSG("Timer already started!");
02069   }
02070
02071   /* Stop timer... */
02072   else if (mode == 2) {
02073     if (starttime[id] > 0) {
02074       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02075       starttime[id] = -1;
02076     }
02077   }
02078
02079   /* Print timer... */
02080   else if (mode == 3)
02081     printf("%s = %.3f s\n", name, runtime[id]);
02082 }
```

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

Write atmospheric data.

Definition at line 2086 of file libtrac.c.

```
02090             {
02091
02092   FILE *in, *out;
02093
02094   char line[LEN];
02095
02096   double r, t0, t1;
02097
02098   int ip, iq, year, mon, day, hour, min, sec;
02099
02100   /* Set time interval for output... */
02101   t0 = t - 0.5 * ctl->dt_mod;
02102   t1 = t + 0.5 * ctl->dt_mod;
02103
02104   /* Write info... */
02105   printf("Write atmospheric data: %s\n", filename);
02106
02107   /* Write ASCII data... */
02108   if (ctl->atm_type == 0) {
02109
02110     /* Check if gnuplot output is requested... */
02111     if (ctl->atm_gpfile[0] != '-') {
02112
02113       /* Create gnuplot pipe... */
02114       if (!(out = popen("gnuplot", "w")))
02115         ERRMSG("Cannot create pipe to gnuplot!");
02116
02117       /* Set plot filename... */
02118       fprintf(out, "set out \"%s.png\"\n", filename);
02119
02120       /* Set time string... */
02121       jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02122       fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02123               year, mon, day, hour, min);
02124
02125       /* Dump gnuplot file to pipe... */
02126       if (!(in = fopen(ctl->atm_gpfile, "r")))
02127         ERRMSG("Cannot open file!");
02128       while (fgets(line, LEN, in))
02129         fprintf(out, "%s", line);
02130       fclose(in);
02131     }
02132
02133     else {
02134
02135       /* Create file... */
02136       if (!(out = fopen(filename, "w")))
02137         ERRMSG("Cannot create file!");
02138     }
02139
02140     /* Write header... */
02141     fprintf(out,
02142             "# $1 = time [s]\n"
02143             "# $2 = altitude [km]\n"
```

```
02144             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02145     for (iq = 0; iq < ctl->nq; iq++)
02146       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02147               ctl->qnt_unit[iq]);
02148     fprintf(out, "\n");
02149
02150     /* Write data... */
02151     for (ip = 0; ip < atm->np; ip++) {
02152
02153       /* Check time... */
02154       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02155         continue;
02156
02157       /* Write output... */
02158       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02159               atm->lon[ip], atm->lat[ip]);
02160       for (iq = 0; iq < ctl->nq; iq++) {
02161         fprintf(out, " ");
02162         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02163       }
02164       fprintf(out, "\n");
02165     }
02166
02167     /* Close file... */
02168     fclose(out);
02169   }
02170
02171   /* Write binary data... */
02172   else if (ctl->atm_type == 1) {
02173
02174     /* Create file... */
02175     if (!(out = fopen(filename, "w")))
02176       ERRMSG("Cannot create file!");
02177
02178     /* Write data... */
02179     FWRITE(&atm->np, int,
02180            1,
02181            out);
02182     FWRITE(atm->time, double,
02183              (size_t) atm->np,
02184            out);
02185     FWRITE(atm->p, double,
02186              (size_t) atm->np,
02187            out);
02188     FWRITE(atm->lon, double,
02189              (size_t) atm->np,
02190            out);
02191     FWRITE(atm->lat, double,
02192              (size_t) atm->np,
02193            out);
02194     for (iq = 0; iq < ctl->nq; iq++)
02195       FWRITE(atm->q[iq], double,
02196                (size_t) atm->np,
02197              out);
02198
02199     /* Close file... */
02200     fclose(out);
02201   }
02202
02203   /* Error... */
02204   else
02205     ERRMSG("Atmospheric data type not supported!");
02206 }
```

Here is the call graph for this function:

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

Write CSI data.

Definition at line 2210 of file libtrac.c.

```
02214           {
02215
02216   static FILE *in, *out;
02217
02218   static char line[LEN];
02219
02220   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02221     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02222
02223   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02224
02225   /* Init... */
02226   if (!init) {
02227     init = 1;
02228
02229     /* Check quantity index for mass... */
02230     if (ctl->qnt_m < 0)
02231       ERRMSG("Need quantity mass to analyze CSI!");
02232
02233     /* Open observation data file... */
02234     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02235     if (!(in = fopen(ctl->csi_obsfile, "r")))
02236       ERRMSG("Cannot open file!");
02237
02238     /* Create new file... */
02239     printf("Write CSI data: %s\n", filename);
02240     if (!(out = fopen(filename, "w")))
02241       ERRMSG("Cannot create file!");
02242
02243     /* Write header... */
02244     fprintf(out,
02245             "# $1 = time [s]\n"
02246             "# $2 = number of hits (cx)\n"
02247             "# $3 = number of misses (cy)\n"
02248             "# $4 = number of false alarms (cz)\n"
02249             "# $5 = number of observations (cx + cy)\n"
02250             "# $6 = number of forecasts (cx + cz)\n"
02251             "# $7 = bias (forecasts/observations) [%%]\n"
02252             "# $8 = probability of detection (POD) [%%]\n"
02253             "# $9 = false alarm rate (FAR) [%%]\n"
02254             "# $10 = critical success index (CSI) [%%]\n\n");
02255   }
02256
02257   /* Set time interval... */
02258   t0 = t - 0.5 * ctl->dt_mod;
02259   t1 = t + 0.5 * ctl->dt_mod;
02260
02261   /* Initialize grid cells... */
02262   for (ix = 0; ix < ctl->csi_nx; ix++)
02263     for (iy = 0; iy < ctl->csi_ny; iy++)
02264       for (iz = 0; iz < ctl->csi_nz; iz++)
02265         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02266
02267   /* Read data... */
02268   while (fgets(line, LEN, in)) {
02269
02270     /* Read data... */
02271     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02272         5)
02273       continue;
02274
02275     /* Check time... */
02276     if (rt < t0)
02277       continue;
02278     if (rt > t1)
02279       break;
02280
02281     /* Calculate indices... */
02282     ix = (int) ((rlon - ctl->csi_lon0)
02283                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02284     iy = (int) ((rlat - ctl->csi_lat0)
02285                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02286     iz = (int) ((rz - ctl->csi_z0)
02287                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02288
02289     /* Check indices... */
02290     if (ix < 0 || ix >= ctl->csi_nx ||
02291         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02292       continue;
02293
02294     /* Get mean observation index... */
02295     obsmean[ix][iy][iz] += robs;
02296     obscount[ix][iy][iz]++;
02297   }
```

```
02298
02299   /* Analyze model data... */
02300   for (ip = 0; ip < atm->np; ip++) {
02301
02302     /* Check time... */
02303     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02304       continue;
02305
02306     /* Get indices... */
02307     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02308                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02309     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02310                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02311     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02312                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02313
02314     /* Check indices... */
02315     if (ix < 0 || ix >= ctl->csi_nx ||
02316         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02317       continue;
02318
02319     /* Get total mass in grid cell... */
02320     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02321   }
02322
02323   /* Analyze all grid cells... */
02324   for (ix = 0; ix < ctl->csi_nx; ix++)
02325     for (iy = 0; iy < ctl->csi_ny; iy++)
02326       for (iz = 0; iz < ctl->csi_nz; iz++) {
02327
02328         /* Calculate mean observation index... */
02329         if (obscount[ix][iy][iz] > 0)
02330           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02331
02332         /* Calculate column density... */
02333         if (modmean[ix][iy][iz] > 0) {
02334           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02335           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02336           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02337           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02338             * cos(lat * M_PI / 180.);
02339           modmean[ix][iy][iz] /= (1e6 * area);
02340         }
02341
02342         /* Calculate CSI... */
02343         if (obscount[ix][iy][iz] > 0) {
02344           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02345               modmean[ix][iy][iz] >= ctl->csi_modmin)
02346             cx++;
02347           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02348                    modmean[ix][iy][iz] < ctl->csi_modmin)
02349             cy++;
02350           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02351                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02352             cz++;
02353         }
02354       }
02355
02356   /* Write output... */
02357   if (fmod(t, ctl->csi_dt_out) == 0) {
02358
02359     /* Write... */
02360     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02361             t, cx, cy, cz, cx + cy, cx + cz,
02362             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02363             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02364             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02365             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02366
02367     /* Set counters to zero... */
02368     cx = cy = cz = 0;
02369   }
02370
02371   /* Close file... */
02372   if (t == ctl->t_stop)
02373     fclose(out);
02374 }
```

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

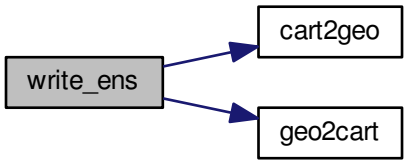
Write ensemble data.

Definition at line 2378 of file libtrac.c.

```
02382           {
02383
02384   static FILE *out;
02385
02386   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02387     t0, t1, x[NENS][3], xm[3];
02388
02389   static int init, ip, iq;
02390
02391   static size_t i, n;
02392
02393   /* Init... */
02394   if (!init) {
02395     init = 1;
02396
02397     /* Check quantities... */
02398     if (ctl->qnt_ens < 0)
02399       ERRMSG("Missing ensemble IDs!");
02400
02401     /* Create new file... */
02402     printf("Write ensemble data: %s\n", filename);
02403     if (!(out = fopen(filename, "w")))
02404       ERRMSG("Cannot create file!");
02405
02406     /* Write header... */
02407     fprintf(out,
02408             "# $1 = time [s]\n"
02409             "# $2 = altitude [km]\n"
02410             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02411     for (iq = 0; iq < ctl->nq; iq++)
02412       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02413               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02414     for (iq = 0; iq < ctl->nq; iq++)
02415       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02416               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02417     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02418   }
02419
02420   /* Set time interval... */
02421   t0 = t - 0.5 * ctl->dt_mod;
02422   t1 = t + 0.5 * ctl->dt_mod;
02423
02424   /* Init... */
02425   ens = GSL_NAN;
02426   n = 0;
02427
02428   /* Loop over air parcels... */
02429   for (ip = 0; ip < atm->np; ip++) {
02430
02431     /* Check time... */
02432     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02433       continue;
02434
02435     /* Check ensemble id... */
02436     if (atm->q[ctl->qnt_ens][ip] != ens) {
02437
02438       /* Write results... */
02439       if (n > 0) {
02440
02441         /* Get mean position... */
02442         xm[0] = xm[1] = xm[2] = 0;
02443         for (i = 0; i < n; i++) {
02444           xm[0] += x[i][0] / (double) n;
02445           xm[1] += x[i][1] / (double) n;
02446           xm[2] += x[i][2] / (double) n;
02447         }
02448         cart2geo(xm, &dummy, &lon, &lat);
02449         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02450                 lat);
02451
02452         /* Get quantity statistics... */
02453         for (iq = 0; iq < ctl->nq; iq++) {
02454           fprintf(out, " ");
02455           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02456         }
02457         for (iq = 0; iq < ctl->nq; iq++) {
02458           fprintf(out, " ");
02459           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02460         }
02461         fprintf(out, " %lu\n", n);
02462       }
02463
02464       /* Init new ensemble... */
02465       ens = atm->q[ctl->qnt_ens][ip];
02466       n = 0;
02467     }
02468
```

```
02469     /* Save data... */
02470     p[n] = atm->p[ip];
02471     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02472     for (iq = 0; iq < ctl->nq; iq++)
02473       q[iq][n] = atm->q[iq][ip];
02474     if ((++n) >= NENS)
02475       ERRMSG("Too many data points!");
02476   }
02477
02478   /* Write results... */
02479   if (n > 0) {
02480
02481     /* Get mean position... */
02482     xm[0] = xm[1] = xm[2] = 0;
02483     for (i = 0; i < n; i++) {
02484       xm[0] += x[i][0] / (double) n;
02485       xm[1] += x[i][1] / (double) n;
02486       xm[2] += x[i][2] / (double) n;
02487     }
02488     cart2geo(xm, &dummy, &lon, &lat);
02489     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02490
02491     /* Get quantity statistics... */
02492     for (iq = 0; iq < ctl->nq; iq++) {
02493       fprintf(out, " ");
02494       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02495     }
02496     for (iq = 0; iq < ctl->nq; iq++) {
02497       fprintf(out, " ");
02498       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02499     }
02500     fprintf(out, " %lu\n", n);
02501   }
02502
02503   /* Close file... */
02504   if (t == ctl->t_stop)
02505     fclose(out);
02506 }
```

Here is the call graph for this function:



5.15.2.35 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 2510 of file libtrac.c.

```
02516             {
02517
02518   FILE *in, *out;
02519
02520   char line[LEN];
02521
02522   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02523     area, rho_air, press, temp, cd, mmr, t0, t1, r;
02524
02525   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02526
```

```
02527   /* Check dimensions... */
02528   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02529     ERRMSG("Grid dimensions too large!");
02530
02531   /* Check quantity index for mass... */
02532   if (ctl->qnt_m < 0)
02533     ERRMSG("Need quantity mass to write grid data!");
02534
02535   /* Set time interval for output... */
02536   t0 = t - 0.5 * ctl->dt_mod;
02537   t1 = t + 0.5 * ctl->dt_mod;
02538
02539   /* Set grid box size... */
02540   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02541   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02542   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02543
02544   /* Initialize grid... */
02545   for (ix = 0; ix < ctl->grid_nx; ix++)
02546     for (iy = 0; iy < ctl->grid_ny; iy++)
02547       for (iz = 0; iz < ctl->grid_nz; iz++)
02548         grid_m[ix][iy][iz] = 0;
02549
02550   /* Average data... */
02551   for (ip = 0; ip < atm->np; ip++)
02552     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02553
02554       /* Get index... */
02555       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02556       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02557       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02558
02559       /* Check indices... */
02560       if (ix < 0 || ix >= ctl->grid_nx ||
02561           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02562         continue;
02563
02564       /* Add mass... */
02565       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02566     }
02567
02568   /* Check if gnuplot output is requested... */
02569   if (ctl->grid_gpfile[0] != '-') {
02570
02571     /* Write info... */
02572     printf("Plot grid data: %s.png\n", filename);
02573
02574     /* Create gnuplot pipe... */
02575     if (!(out = popen("gnuplot", "w")))
02576       ERRMSG("Cannot create pipe to gnuplot!");
02577
02578     /* Set plot filename... */
02579     fprintf(out, "set out \"%s.png\"\n", filename);
02580
02581     /* Set time string... */
02582     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02583     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02584             year, mon, day, hour, min);
02585
02586     /* Dump gnuplot file to pipe... */
02587     if (!(in = fopen(ctl->grid_gpfile, "r")))
02588       ERRMSG("Cannot open file!");
02589     while (fgets(line, LEN, in))
02590       fprintf(out, "%s", line);
02591     fclose(in);
02592   }
02593
02594   else {
02595
02596     /* Write info... */
02597     printf("Write grid data: %s\n", filename);
02598
02599     /* Create file... */
02600     if (!(out = fopen(filename, "w")))
02601       ERRMSG("Cannot create file!");
02602   }
02603
02604   /* Write header... */
02605   fprintf(out,
02606           "# $1 = time [s]\n"
02607           "# $2 = altitude [km]\n"
02608           "# $3 = longitude [deg]\n"
02609           "# $4 = latitude [deg]\n"
02610           "# $5 = surface area [km^2]\n"
02611           "# $6 = layer width [km]\n"
02612           "# $7 = temperature [K]\n"
02613           "# $8 = column density [kg/m^2]\n"
```

```
02614           "# $9 = mass mixing ratio [1]\n\n");
02615
02616   /* Write data... */
02617   for (ix = 0; ix < ctl->grid_nx; ix++) {
02618     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02619       fprintf(out, "\n");
02620     for (iy = 0; iy < ctl->grid_ny; iy++) {
02621       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02622         fprintf(out, "\n");
02623       for (iz = 0; iz < ctl->grid_nz; iz++)
02624         if (!ctl->grid_sparse
02625             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
02626
02627           /* Set coordinates... */
02628           z = ctl->grid_z0 + dz * (iz + 0.5);
02629           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02630           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02631
02632           /* Get pressure and temperature... */
02633           press = P(z);
02634           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02635                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
02636
02637           /* Calculate surface area... */
02638           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
02639             * cos(lat * M_PI / 180.);
02640
02641           /* Calculate column density... */
02642           cd = grid_m[ix][iy][iz] / (1e6 * area);
02643
02644           /* Calculate mass mixing ratio... */
02645           rho_air = 100. * press / (R0 * temp);
02646           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
02647
02648           /* Write output... */
02649           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
02650                   t, z, lon, lat, area, dz, temp, cd, mmr);
02651         }
02652     }
02653   }
02654
02655   /* Close file... */
02656   fclose(out);
02657 }
```

Here is the call graph for this function:

5.15.2.36 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 2661 of file libtrac.c.

```
02667           {
02668
02669   static FILE *in, *out;
02670
02671   static char line[LEN];
02672
02673   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
02674     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
02675     press, temp, rho_air, mmr, h2o, o3;
02676
02677   static int init, obscount[GX][GY], ip, ix, iy, iz;
02678
02679   /* Init... */
02680   if (!init) {
02681     init = 1;
02682
02683     /* Check quantity index for mass... */
02684     if (ctl->qnt_m < 0)
02685       ERRMSG("Need quantity mass!");
02686
02687     /* Check dimensions... */
02688     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02689       ERRMSG("Grid dimensions too large!");
02690
02691     /* Open observation data file... */
02692     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
```

```
02693     if (!(in = fopen(ctl->prof_obsfile, "r")))
02694       ERRMSG("Cannot open file!");
02695
02696     /* Create new file... */
02697     printf("Write profile data: %s\n", filename);
02698     if (!(out = fopen(filename, "w")))
02699       ERRMSG("Cannot create file!");
02700
02701     /* Write header... */
02702     fprintf(out,
02703             "# $1  = time [s]\n"
02704             "# $2  = altitude [km]\n"
02705             "# $3  = longitude [deg]\n"
02706             "# $4  = latitude [deg]\n"
02707             "# $5  = pressure [hPa]\n"
02708             "# $6  = temperature [K]\n"
02709             "# $7  = mass mixing ratio [1]\n"
02710             "# $8  = H2O volume mixing ratio [1]\n"
02711             "# $9  = O3 volume mixing ratio [1]\n"
02712             "# $10 = mean BT index [K]\n");
02713
02714     /* Set grid box size... */
02715     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
02716     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02717     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02718   }
02719
02720   /* Set time interval... */
02721   t0 = t - 0.5 * ctl->dt_mod;
02722   t1 = t + 0.5 * ctl->dt_mod;
02723
02724   /* Initialize... */
02725   for (ix = 0; ix < ctl->prof_nx; ix++)
02726     for (iy = 0; iy < ctl->prof_ny; iy++) {
02727       obsmean[ix][iy] = 0;
02728       obscount[ix][iy] = 0;
02729       tmean[ix][iy] = 0;
02730       for (iz = 0; iz < ctl->prof_nz; iz++)
02731         mass[ix][iy][iz] = 0;
02732     }
02733
02734   /* Read data... */
02735   while (fgets(line, LEN, in)) {
02736
02737     /* Read data... */
02738     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
02739       continue;
02740
02741     /* Check time... */
02742     if (rt < t0)
02743       continue;
02744     if (rt > t1)
02745       break;
02746
02747     /* Calculate indices... */
02748     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
02749     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02750
02751     /* Check indices... */
02752     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02753       continue;
02754
02755     /* Get mean observation index... */
02756     obsmean[ix][iy] += robs;
02757     tmean[ix][iy] += rt;
02758     obscount[ix][iy]++;
02759   }
02760
02761   /* Analyze model data... */
02762   for (ip = 0; ip < atm->np; ip++) {
02763
02764     /* Check time... */
02765     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02766       continue;
02767
02768     /* Get indices... */
02769     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
02770     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02771     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02772
02773     /* Check indices... */
02774     if (ix < 0 || ix >= ctl->prof_nx ||
02775         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02776       continue;
02777
02778     /* Get total mass in grid cell... */
02779     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
```

```
02780   }
02781
02782   /* Extract profiles... */
02783   for (ix = 0; ix < ctl->prof_nx; ix++)
02784     for (iy = 0; iy < ctl->prof_ny; iy++)
02785       if (obscount[ix][iy] > 0) {
02786
02787         /* Write output... */
02788         fprintf(out, "\n");
02789
02790         /* Loop over altitudes... */
02791         for (iz = 0; iz < ctl->prof_nz; iz++) {
02792
02793           /* Set coordinates... */
02794           z = ctl->prof_z0 + dz * (iz + 0.5);
02795           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
02796           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
02797
02798           /* Get meteorological data... */
02799           press = P(z);
02800           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02801                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02802
02803           /* Calculate mass mixing ratio... */
02804           rho_air = 100. * press / (R0 * temp);
02805           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
02806             * cos(lat * M_PI / 180.);
02807           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02808
02809           /* Write output... */
02810           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02811                   tmean[ix][iy] / obscount[ix][iy],
02812                   z, lon, lat, press, temp, mmr, h2o, o3,
02813                   obsmean[ix][iy] / obscount[ix][iy]);
02814         }
02815       }
02816
02817   /* Close file... */
02818   if (t == ctl->t_stop)
02819     fclose(out);
02820 }
```

Here is the call graph for this function:

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

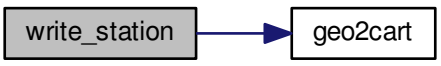
Write station data.

Definition at line 2824 of file libtrac.c.

```
02828             {
02829
02830   static FILE *out;
02831
02832   static double rmax2, t0, t1, x0[3], x1[3];
02833
02834   static int init, ip, iq;
02835
02836   /* Init... */
02837   if (!init) {
02838     init = 1;
02839
02840     /* Write info... */
02841     printf("Write station data: %s\n", filename);
02842
02843     /* Create new file... */
02844     if (!(out = fopen(filename, "w")))
02845       ERRMSG("Cannot create file!");
02846
02847     /* Write header... */
02848     fprintf(out,
02849             "# $1 = time [s]\n"
02850             "# $2 = altitude [km]\n"
02851             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02852     for (iq = 0; iq < ctl->nq; iq++)
02853       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
02854               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02855     fprintf(out, "\n");
02856
```

```
02857     /* Set geolocation and search radius... */
02858     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02859     rmax2 = gsl_pow_2(ctl->stat_r);
02860   }
02861
02862   /* Set time interval for output... */
02863   t0 = t - 0.5 * ctl->dt_mod;
02864   t1 = t + 0.5 * ctl->dt_mod;
02865
02866   /* Loop over air parcels... */
02867   for (ip = 0; ip < atm->np; ip++) {
02868
02869     /* Check time... */
02870     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02871       continue;
02872
02873     /* Check station flag... */
02874     if (ctl->qnt_stat >= 0)
02875       if (atm->q[ctl->qnt_stat][ip])
02876         continue;
02877
02878     /* Get Cartesian coordinates... */
02879     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02880
02881     /* Check horizontal distance... */
02882     if (DIST2(x0, x1) > rmax2)
02883       continue;
02884
02885     /* Set station flag... */
02886     if (ctl->qnt_stat >= 0)
02887       atm->q[ctl->qnt_stat][ip] = 1;
02888
02889     /* Write data... */
02890     fprintf(out, "%.2f %g %g %g",
02891             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02892     for (iq = 0; iq < ctl->nq; iq++) {
02893       fprintf(out, " ");
02894       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02895     }
02896     fprintf(out, "\n");
02897   }
02898
02899   /* Close file... */
02900   if (t == ctl->t_stop)
02901     fclose(out);
02902 }
```

Here is the call graph for this function:



5.16 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   Copyright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*****************************************************************************/
00028
00029 void cart2geo(
00030   double *x,
00031   double *z,
00032   double *lon,
00033   double *lat) {
00034
00035   double radius;
00036
00037   radius = NORM(x);
00038   *lat = asin(x[2] / radius) * 180 / M_PI;
00039   *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040   *z = radius - RE;
00041 }
00042
00043 /*****************************************************************************/
00044
00045 double clim_hno3(
00046   double t,
00047   double lat,
00048   double p) {
00049
00050   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051     9072000.00, 11664000.00, 14342400.00,
00052     16934400.00, 19612800.00, 22291200.00,
00053     24883200.00, 27561600.00, 30153600.00
00054   };
00055
00056   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00057     5, 15, 25, 35, 45, 55, 65, 75, 85
00058   };
00059
00060   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00061     31.6228, 46.4159, 68.1292, 100, 146.78
00062   };
00063
00064   static double hno3[12][18][10] = {
00065     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00066      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00067      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00068      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00069      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00070      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00071      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00072      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00073      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00074      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00075      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00076      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00077      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00078      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00079      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00080      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00081      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00082      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00083     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00084      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00085      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00086      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00087      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00088      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00089      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00090      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00091      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00092      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00093      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00094      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00095      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00096      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00097      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00098      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00099      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00100      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00101     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00102      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00103      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00104      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00105      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00106      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
```

```
00107       {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00108       {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00109       {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00110       {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00111       {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00112       {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00113       {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00114       {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00115       {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00116       {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00117       {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00118       {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00119      {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00120       {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00121       {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00122       {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00123       {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00124       {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00125       {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00126       {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00127       {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00128       {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00129       {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00130       {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00131       {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00132       {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00133       {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00134       {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00135       {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00136       {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00137      {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00138       {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00139       {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00140       {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00141       {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00142       {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00143       {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00144       {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00145       {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00146       {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00147       {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00148       {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00149       {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00150       {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00151       {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00152       {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00153       {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00154       {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00155      {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00156       {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00157       {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00158       {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00159       {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00160       {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00161       {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00162       {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00163       {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00164       {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00165       {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00166       {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00167       {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00168       {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00169       {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00170       {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00171       {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00172       {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00173      {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00174       {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00175       {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00176       {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00177       {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00178       {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00179       {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00180       {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00181       {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00182       {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00183       {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00184       {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00185       {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00186       {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00187       {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00188       {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00189       {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00190       {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00191      {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00192       {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00193       {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
```

```
00194      {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00195      {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00196      {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00197      {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00198      {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00199      {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00200      {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00201      {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00202      {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00203      {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00204      {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00205      {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00206      {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00207      {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00208      {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00209     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00210      {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00211      {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00212      {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00213      {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00214      {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00215      {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00216      {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00217      {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00218      {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00219      {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00220      {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00221      {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00222      {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00223      {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00224      {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00225      {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00226      {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00227     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00228      {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00229      {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00230      {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00231      {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00232      {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00233      {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00234      {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00235      {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00236      {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00237      {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00238      {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00239      {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00240      {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00241      {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00242      {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00243      {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00244      {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00245     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00246      {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00247      {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00248      {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00249      {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00250      {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00251      {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00252      {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00253      {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00254      {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00255      {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00256      {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00257      {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00258      {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00259      {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00260      {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00261      {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00262      {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00263     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00264      {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00265      {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00266      {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00267      {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00268      {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00269      {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00270      {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00271      {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00272      {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00273      {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00274      {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00275      {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00276      {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00277      {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00278      {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00279      {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00280      {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
```

```
00281   };
00282
00283   double aux00, aux01, aux10, aux11, sec;
00284
00285   int ilat, ip, isec;
00286
00287   /* Get seconds since begin of year... */
00288   sec = fmod(t, 365.25 * 86400.);
00289
00290   /* Get indices... */
00291   ilat = locate(lats, 18, lat);
00292   ip = locate(ps, 10, p);
00293   isec = locate(secs, 12, sec);
00294
00295   /* Interpolate... */
00296   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
00297               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
00298   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
00299               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
00300   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
00301               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
00302   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
00303               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
00304   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
00305   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
00306   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
00307 }
00308
00309 /*****************************************************************************/
00310
00311 double clim_tropo(
00312   double t,
00313   double lat) {
00314
00315   static double doys[12]
00316   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
00318   static double lats[73]
00319     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00320     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00321     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00322     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00323     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00324     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00325     75, 77.5, 80, 82.5, 85, 87.5, 90
00326   };
00327
00328   static double tps[12][73]
00329     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00330          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00331          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00332          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00333          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00334          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00335          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00336          275.3, 275.6, 275.4, 274.1, 273.5},
00337     {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00338      300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00339      150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00340      98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00341      98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00342      220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00343      284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00344      287.5, 286.2, 285.8},
00345     {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00346      297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00347      161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00348      100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00349      99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00350      186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00351      279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00352      304.3, 304.9, 306, 306.6, 306.2, 306},
00353     {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00354      290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00355      195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00356      102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00357      99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00358      148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00359      263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00360      315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00361     {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00362      260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00363      205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00364      101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00365      102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00366      165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00367      273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
```

```
00368    325.3, 325.8, 325.8},
00369   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00370    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00371    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00372    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00373    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00374    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00375    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00376    308.5, 312.2, 313.1, 313.3},
00377   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00378    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00379    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00380    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00381    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00382    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00383    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00384    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00385   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00386    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00387    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00388    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00389    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00390    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00391    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00392    278.2, 282.6, 287.4, 290.9, 292.5, 293},
00393   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00394    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00395    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00396    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00397    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00398    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00399    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00400    276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00401   {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00402    215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00403    237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00404    111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00405    106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00406    112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00407    206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00408    279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00409    305.1},
00410   {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00411    253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00412    223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00413    108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00414    102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00415    109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00416    241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00417    286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00418   {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00419    284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00420    175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00421    100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00422    100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00423    186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00424    280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00425    281.7, 281.1, 281.2}
00426   };
00427
00428   double doy, p0, p1, pt;
00429
00430   int imon, ilat;
00431
00432   /* Get day of year... */
00433   doy = fmod(t / 86400., 365.25);
00434   while (doy < 0)
00435     doy += 365.25;
00436
00437   /* Get indices... */
00438   imon = locate(doys, 12, doy);
00439   ilat = locate(lats, 73, lat);
00440
00441   /* Get tropopause pressure... */
00442   p0 = LIN(lats[ilat], tps[imon][ilat],
00443            lats[ilat + 1], tps[imon][ilat + 1], lat);
00444   p1 = LIN(lats[ilat], tps[imon + 1][ilat],
00445            lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
00446   pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447
00448   /* Return tropopause pressure... */
00449   return pt;
00450 }
00451
00452 /*****************************************************************************/
00453
00454 double deg2dx(
```

```
00455   double dlon,
00456   double lat) {
00457
00458   return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00459 }
00460
00461 /*****************************************************************************/
00462
00463 double deg2dy(
00464   double dlat) {
00465
00466   return dlat * M_PI * RE / 180.;
00467 }
00468
00469 /*****************************************************************************/
00470
00471 double dp2dz(
00472   double dp,
00473   double p) {
00474
00475   return -dp * H0 / p;
00476 }
00477
00478 /*****************************************************************************/
00479
00480 double dx2deg(
00481   double dx,
00482   double lat) {
00483
00484   /* Avoid singularity at poles... */
00485   if (lat < -89.999 || lat > 89.999)
00486     return 0;
00487   else
00488     return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00489 }
00490
00491 /*****************************************************************************/
00492
00493 double dy2deg(
00494   double dy) {
00495
00496   return dy * 180. / (M_PI * RE);
00497 }
00498
00499 /*****************************************************************************/
00500
00501 double dz2dp(
00502   double dz,
00503   double p) {
00504
00505   return -dz * p / H0;
00506 }
00507
00508 /*****************************************************************************/
00509
00510 void geo2cart(
00511   double z,
00512   double lon,
00513   double lat,
00514   double *x) {
00515
00516   double radius;
00517
00518   radius = z + RE;
00519   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00520   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00521   x[2] = radius * sin(lat / 180 * M_PI);
00522 }
00523
00524 /*****************************************************************************/
00525
00526 void get_met(
00527   ctl_t * ctl,
00528   char *metbase,
00529   double t,
00530   met_t * met0,
00531   met_t * met1) {
00532
00533   char filename[LEN];
00534
00535   static int init;
00536
00537   /* Init... */
00538   if (!init) {
00539     init = 1;
00540
00541     get_met_help(t, -1, metbase, ctl->dt_met, filename);
```

```
00542     read_met(ctl, filename, met0);
00543
00544     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00545     read_met(ctl, filename, met1);
00546   }
00547
00548   /* Read new data for forward trajectories... */
00549   if (t > met1->time && ctl->direction == 1) {
00550     memcpy(met0, met1, sizeof(met_t));
00551     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00552     read_met(ctl, filename, met1);
00553   }
00554
00555   /* Read new data for backward trajectories... */
00556   if (t < met0->time && ctl->direction == -1) {
00557     memcpy(met1, met0, sizeof(met_t));
00558     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00559     read_met(ctl, filename, met0);
00560   }
00561 }
00562
00563 /*****************************************************************************/
00564
00565 void get_met_help(
00566   double t,
00567   int direct,
00568   char *metbase,
00569   double dt_met,
00570   char *filename) {
00571
00572   double t6, r;
00573
00574   int year, mon, day, hour, min, sec;
00575
00576   /* Round time to fixed intervals... */
00577   if (direct == -1)
00578     t6 = floor(t / dt_met) * dt_met;
00579   else
00580     t6 = ceil(t / dt_met) * dt_met;
00581
00582   /* Decode time... */
00583   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00584
00585   /* Set filename... */
00586   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00587 }
00588
00589 /*****************************************************************************/
00590
00591 void intpol_met_2d(
00592   double array[EX][EY],
00593   int ix,
00594   int iy,
00595   double wx,
00596   double wy,
00597   double *var) {
00598
00599   double aux00, aux01, aux10, aux11;
00600
00601   /* Set variables... */
00602   aux00 = array[ix][iy];
00603   aux01 = array[ix][iy + 1];
00604   aux10 = array[ix + 1][iy];
00605   aux11 = array[ix + 1][iy + 1];
00606
00607   /* Interpolate horizontally... */
00608   aux00 = wy * (aux00 - aux01) + aux01;
00609   aux11 = wy * (aux10 - aux11) + aux11;
00610   *var = wx * (aux00 - aux11) + aux11;
00611 }
00612
00613 /*****************************************************************************/
00614
00615 void intpol_met_3d(
00616   float array[EX][EY][EP],
00617   int ip,
00618   int ix,
00619   int iy,
00620   double wp,
00621   double wx,
00622   double wy,
00623   double *var) {
00624
00625   double aux00, aux01, aux10, aux11;
00626
00627   /* Interpolate vertically... */
```

```
00628   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00629     + array[ix][iy][ip + 1];
00630   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00631     + array[ix][iy + 1][ip + 1];
00632   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00633     + array[ix + 1][iy][ip + 1];
00634   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00635     + array[ix + 1][iy + 1][ip + 1];
00636
00637   /* Interpolate horizontally... */
00638   aux00 = wy * (aux00 - aux01) + aux01;
00639   aux11 = wy * (aux10 - aux11) + aux11;
00640   *var = wx * (aux00 - aux11) + aux11;
00641 }
00642
00643 /*****************************************************************************/
00644
00645 void intpol_met_space(
00646   met_t * met,
00647   double p,
00648   double lon,
00649   double lat,
00650   double *ps,
00651   double *pt,
00652   double *z,
00653   double *t,
00654   double *u,
00655   double *v,
00656   double *w,
00657   double *h2o,
00658   double *o3) {
00659
00660   double wp, wx, wy;
00661
00662   int ip, ix, iy;
00663
00664   /* Check longitude... */
00665   if (met->lon[met->nx - 1] > 180 && lon < 0)
00666     lon += 360;
00667
00668   /* Get indices... */
00669   ip = locate(met->p, met->np, p);
00670   ix = locate(met->lon, met->nx, lon);
00671   iy = locate(met->lat, met->ny, lat);
00672
00673   /* Get weights... */
00674   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00675   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00676   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00677
00678   /* Interpolate... */
00679   if (ps != NULL)
00680     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00681   if (pt != NULL)
00682     intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00683   if (z != NULL)
00684     intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00685   if (t != NULL)
00686     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00687   if (u != NULL)
00688     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00689   if (v != NULL)
00690     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00691   if (w != NULL)
00692     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00693   if (h2o != NULL)
00694     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00695   if (o3 != NULL)
00696     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00697 }
00698
00699 /*****************************************************************************/
00700
00701 void intpol_met_time(
00702   met_t * met0,
00703   met_t * met1,
00704   double ts,
00705   double p,
00706   double lon,
00707   double lat,
00708   double *ps,
00709   double *pt,
00710   double *z,
00711   double *t,
00712   double *u,
00713   double *v,
00714   double *w,
```

```
00715   double *h2o,
00716   double *o3) {
00717
00718   double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, t0, t1, u0, u1, v0, v1,
00719     w0, w1, wt, z0, z1;
00720
00721   /* Spatial interpolation... */
00722   intpol_met_space(met0, p, lon, lat,
00723                    ps == NULL ? NULL : &ps0,
00724                    pt == NULL ? NULL : &pt0,
00725                    z == NULL ? NULL : &z0,
00726                    t == NULL ? NULL : &t0,
00727                    u == NULL ? NULL : &u0,
00728                    v == NULL ? NULL : &v0,
00729                    w == NULL ? NULL : &w0,
00730                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00731   intpol_met_space(met1, p, lon, lat,
00732                    ps == NULL ? NULL : &ps1,
00733                    pt == NULL ? NULL : &pt1,
00734                    z == NULL ? NULL : &z1,
00735                    t == NULL ? NULL : &t1,
00736                    u == NULL ? NULL : &u1,
00737                    v == NULL ? NULL : &v1,
00738                    w == NULL ? NULL : &w1,
00739                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00740
00741   /* Get weighting factor... */
00742   wt = (met1->time - ts) / (met1->time - met0->time);
00743
00744   /* Interpolate... */
00745   if (ps != NULL)
00746     *ps = wt * (ps0 - ps1) + ps1;
00747   if (pt != NULL)
00748     *pt = wt * (pt0 - pt1) + pt1;
00749   if (z != NULL)
00750     *z = wt * (z0 - z1) + z1;
00751   if (t != NULL)
00752     *t = wt * (t0 - t1) + t1;
00753   if (u != NULL)
00754     *u = wt * (u0 - u1) + u1;
00755   if (v != NULL)
00756     *v = wt * (v0 - v1) + v1;
00757   if (w != NULL)
00758     *w = wt * (w0 - w1) + w1;
00759   if (h2o != NULL)
00760     *h2o = wt * (h2o0 - h2o1) + h2o1;
00761   if (o3 != NULL)
00762     *o3 = wt * (o30 - o31) + o31;
00763 }
00764
00765 /*****************************************************************************/
00766
00767 void jsec2time(
00768   double jsec,
00769   int *year,
00770   int *mon,
00771   int *day,
00772   int *hour,
00773   int *min,
00774   int *sec,
00775   double *remain) {
00776
00777   struct tm t0, *t1;
00778
00779   time_t jsec0;
00780
00781   t0.tm_year = 100;
00782   t0.tm_mon = 0;
00783   t0.tm_mday = 1;
00784   t0.tm_hour = 0;
00785   t0.tm_min = 0;
00786   t0.tm_sec = 0;
00787
00788   jsec0 = (time_t) jsec + timegm(&t0);
00789   t1 = gmtime(&jsec0);
00790
00791   *year = t1->tm_year + 1900;
00792   *mon = t1->tm_mon + 1;
00793   *day = t1->tm_mday;
00794   *hour = t1->tm_hour;
00795   *min = t1->tm_min;
00796   *sec = t1->tm_sec;
00797   *remain = jsec - floor(jsec);
00798 }
00799
00800 /*****************************************************************************/
00801
```

```
00802 int locate(
00803   double *xx,
00804   int n,
00805   double x) {
00806
00807   int i, ilo, ihi;
00808
00809   ilo = 0;
00810   ihi = n - 1;
00811   i = (ihi + ilo) >> 1;
00812
00813   if (xx[i] < xx[i + 1])
00814     while (ihi > ilo + 1) {
00815       i = (ihi + ilo) >> 1;
00816       if (xx[i] > x)
00817         ihi = i;
00818       else
00819         ilo = i;
00820   } else
00821     while (ihi > ilo + 1) {
00822       i = (ihi + ilo) >> 1;
00823       if (xx[i] <= x)
00824         ihi = i;
00825       else
00826         ilo = i;
00827     }
00828
00829   return ilo;
00830 }
00831
00832 /*****************************************************************************/
00833
00834 void read_atm(
00835   const char *filename,
00836   ctl_t * ctl,
00837   atm_t * atm) {
00838
00839   FILE *in;
00840
00841   char line[LEN], *tok;
00842
00843   double t0;
00844
00845   int dimid, ip, iq, ncid, varid;
00846
00847   size_t nparts;
00848
00849   /* Init... */
00850   atm->np = 0;
00851
00852   /* Write info... */
00853   printf("Read atmospheric data: %s\n", filename);
00854
00855   /* Read ASCII data... */
00856   if (ctl->atm_type == 0) {
00857
00858     /* Open file... */
00859     if (!(in = fopen(filename, "r")))
00860       ERRMSG("Cannot open file!");
00861
00862     /* Read line... */
00863     while (fgets(line, LEN, in)) {
00864
00865       /* Read data... */
00866       TOK(line, tok, "%lg", atm->time[atm->np]);
00867       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00868       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00869       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00870       for (iq = 0; iq < ctl->nq; iq++)
00871         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00872
00873       /* Convert altitude to pressure... */
00874       atm->p[atm->np] = P(atm->p[atm->np]);
00875
00876       /* Increment data point counter... */
00877       if ((++atm->np) > NP)
00878         ERRMSG("Too many data points!");
00879     }
00880
00881     /* Close file... */
00882     fclose(in);
00883   }
00884
00885   /* Read binary data... */
00886   else if (ctl->atm_type == 1) {
00887
00888     /* Open file... */
```

```
00889     if (!(in = fopen(filename, "r")))
00890       ERRMSG("Cannot open file!");
00891
00892     /* Read data... */
00893     FREAD(&atm->np, int,
00894           1,
00895           in);
00896     FREAD(atm->time, double,
00897             (size_t) atm->np,
00898           in);
00899     FREAD(atm->p, double,
00900             (size_t) atm->np,
00901           in);
00902     FREAD(atm->lon, double,
00903             (size_t) atm->np,
00904           in);
00905     FREAD(atm->lat, double,
00906             (size_t) atm->np,
00907           in);
00908     for (iq = 0; iq < ctl->nq; iq++)
00909       FREAD(atm->q[iq], double,
00910               (size_t) atm->np,
00911             in);
00912
00913     /* Close file... */
00914     fclose(in);
00915   }
00916
00917   /* Read netCDF data... */
00918   else if (ctl->atm_type == 2) {
00919
00920     /* Open file... */
00921     NC(nc_open(filename, NC_NOWRITE, &ncid));
00922
00923     /* Get dimensions... */
00924     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00925     NC(nc_inq_dimlen(ncid, dimid, &nparts));
00926     atm->np = (int) nparts;
00927     if (atm->np > NP)
00928       ERRMSG("Too many particles!");
00929
00930     /* Get time... */
00931     NC(nc_inq_varid(ncid, "time", &varid));
00932     NC(nc_get_var_double(ncid, varid, &t0));
00933     for (ip = 0; ip < atm->np; ip++)
00934       atm->time[ip] = t0;
00935
00936     /* Read geolocations... */
00937     NC(nc_inq_varid(ncid, "PRESS", &varid));
00938     NC(nc_get_var_double(ncid, varid, atm->p));
00939     NC(nc_inq_varid(ncid, "LON", &varid));
00940     NC(nc_get_var_double(ncid, varid, atm->lon));
00941     NC(nc_inq_varid(ncid, "LAT", &varid));
00942     NC(nc_get_var_double(ncid, varid, atm->lat));
00943
00944     /* Read variables... */
00945     if (ctl->qnt_p >= 0)
00946       if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
00947         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
00948     if (ctl->qnt_t >= 0)
00949       if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
00950         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
00951     if (ctl->qnt_u >= 0)
00952       if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
00953         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
00954     if (ctl->qnt_v >= 0)
00955       if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
00956         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
00957     if (ctl->qnt_w >= 0)
00958       if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
00959         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
00960     if (ctl->qnt_h2o >= 0)
00961       if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
00962         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
00963     if (ctl->qnt_o3 >= 0)
00964       if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
00965         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
00966     if (ctl->qnt_theta >= 0)
00967       if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
00968         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
00969     if (ctl->qnt_pv >= 0)
00970       if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
00971         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
00972
00973     /* Check data... */
00974     for (ip = 0; ip < atm->np; ip++)
00975       if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
```

```
00976           || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
00977           || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
00978           || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
00979           || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
00980         atm->time[ip] = GSL_NAN;
00981         atm->p[ip] = GSL_NAN;
00982         atm->lon[ip] = GSL_NAN;
00983         atm->lat[ip] = GSL_NAN;
00984         for (iq = 0; iq < ctl->nq; iq++)
00985           atm->q[iq][ip] = GSL_NAN;
00986       } else {
00987         if (ctl->qnt_h2o >= 0)
00988           atm->q[ctl->qnt_h2o][ip] *= 1.608;
00989         if (atm->lon[ip] > 180)
00990           atm->lon[ip] -= 360;
00991       }
00992
00993     /* Close file... */
00994     NC(nc_close(ncid));
00995   }
00996
00997   /* Error... */
00998   else
00999     ERRMSG("Atmospheric data type not supported!");
01000
01001   /* Check number of points... */
01002   if (atm->np < 1)
01003     ERRMSG("Can not read any data!");
01004 }
01005
01006 /*****************************************************************************/
01007
01008 void read_ctl(
01009   const char *filename,
01010   int argc,
01011   char *argv[],
01012   ctl_t * ctl) {
01013
01014   int ip, iq;
01015
01016   /* Write info... */
01017   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01018          "(executable: %s | compiled: %s, %s)\n\n",
01019          argv[0], __DATE__, __TIME__);
01020
01021   /* Initialize quantity indices... */
01022   ctl->qnt_ens = -1;
01023   ctl->qnt_m = -1;
01024   ctl->qnt_r = -1;
01025   ctl->qnt_rho = -1;
01026   ctl->qnt_ps = -1;
01027   ctl->qnt_pt = -1;
01028   ctl->qnt_z = -1;
01029   ctl->qnt_p = -1;
01030   ctl->qnt_t = -1;
01031   ctl->qnt_u = -1;
01032   ctl->qnt_v = -1;
01033   ctl->qnt_w = -1;
01034   ctl->qnt_h2o = -1;
01035   ctl->qnt_o3 = -1;
01036   ctl->qnt_theta = -1;
01037   ctl->qnt_pv = -1;
01038   ctl->qnt_tice = -1;
01039   ctl->qnt_tsts = -1;
01040   ctl->qnt_tnat = -1;
01041   ctl->qnt_gw_var = -1;
01042   ctl->qnt_stat = -1;
01043
01044   /* Read quantities... */
01045   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01046   if (ctl->nq > NQ)
01047     ERRMSG("Too many quantities!");
01048   for (iq = 0; iq < ctl->nq; iq++) {
01049
01050     /* Read quantity name and format... */
01051     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01052     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01053              ctl->qnt_format[iq]);
01054
01055     /* Try to identify quantity... */
01056     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01057       ctl->qnt_ens = iq;
01058       sprintf(ctl->qnt_unit[iq], "-");
01059     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01060       ctl->qnt_m = iq;
01061       sprintf(ctl->qnt_unit[iq], "kg");
01062     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
```

```
01063       ctl->qnt_r = iq;
01064       sprintf(ctl->qnt_unit[iq], "m");
01065     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01066       ctl->qnt_rho = iq;
01067       sprintf(ctl->qnt_unit[iq], "kg/m^3");
01068     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01069       ctl->qnt_ps = iq;
01070       sprintf(ctl->qnt_unit[iq], "hPa");
01071     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01072       ctl->qnt_pt = iq;
01073       sprintf(ctl->qnt_unit[iq], "hPa");
01074     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01075       ctl->qnt_z = iq;
01076       sprintf(ctl->qnt_unit[iq], "km");
01077     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01078       ctl->qnt_p = iq;
01079       sprintf(ctl->qnt_unit[iq], "hPa");
01080     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01081       ctl->qnt_t = iq;
01082       sprintf(ctl->qnt_unit[iq], "K");
01083     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01084       ctl->qnt_u = iq;
01085       sprintf(ctl->qnt_unit[iq], "m/s");
01086     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01087       ctl->qnt_v = iq;
01088       sprintf(ctl->qnt_unit[iq], "m/s");
01089     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01090       ctl->qnt_w = iq;
01091       sprintf(ctl->qnt_unit[iq], "hPa/s");
01092     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01093       ctl->qnt_h2o = iq;
01094       sprintf(ctl->qnt_unit[iq], "1");
01095     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01096       ctl->qnt_o3 = iq;
01097       sprintf(ctl->qnt_unit[iq], "1");
01098     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01099       ctl->qnt_theta = iq;
01100       sprintf(ctl->qnt_unit[iq], "K");
01101     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01102       ctl->qnt_pv = iq;
01103       sprintf(ctl->qnt_unit[iq], "PVU");
01104     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01105       ctl->qnt_tice = iq;
01106       sprintf(ctl->qnt_unit[iq], "K");
01107     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01108       ctl->qnt_tsts = iq;
01109       sprintf(ctl->qnt_unit[iq], "K");
01110     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01111       ctl->qnt_tnat = iq;
01112       sprintf(ctl->qnt_unit[iq], "K");
01113     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
01114       ctl->qnt_gw_var = iq;
01115       sprintf(ctl->qnt_unit[iq], "K^2");
01116     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01117       ctl->qnt_stat = iq;
01118       sprintf(ctl->qnt_unit[iq], "-");
01119     } else
01120       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01121   }
01122
01123   /* Time steps of simulation... */
01124   ctl->direction =
01125     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01126   if (ctl->direction != -1 && ctl->direction != 1)
01127     ERRMSG("Set DIRECTION to -1 or 1!");
01128   ctl->t_start =
01129     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
01130   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
01131   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01132
01133   /* Meteorological data... */
01134   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01135   ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01136   ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01137   ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01138   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01139   if (ctl->met_np > EP)
01140     ERRMSG("Too many levels!");
01141   for (ip = 0; ip < ctl->met_np; ip++)
01142     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01143   ctl->met_tropo
01144     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01145   scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01146   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01147
01148   /* Isosurface parameters... */
01149   ctl->isosurf
```

```
01150     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01151   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01152
01153   /* Diffusion parameters... */
01154   ctl->turb_dx_trop
01155     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
01156   ctl->turb_dx_strat
01157     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
01158   ctl->turb_dz_trop
01159     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
01160   ctl->turb_dz_strat
01161     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01162   ctl->turb_meso =
01163     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
01164
01165   /* Life time of particles... */
01166   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01167   ctl->tdec_strat =
01168     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01169
01170   /* PSC analysis... */
01171   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01172   ctl->psc_hno3 =
01173     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01174
01175   /* Gravity wave analysis... */
01176   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
01177
01178   /* Output of atmospheric data... */
01179   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
01180   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01181   ctl->atm_dt_out =
01182     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01183   ctl->atm_filter =
01184     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01185   ctl->atm_type =
01186     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01187
01188   /* Output of CSI data... */
01189   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01190   ctl->csi_dt_out =
01191     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01192   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
01193            ctl->csi_obsfile);
01194   ctl->csi_obsmin =
01195     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01196   ctl->csi_modmin =
01197     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01198   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01199   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01200   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01201   ctl->csi_lon0 =
01202     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01203   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01204   ctl->csi_nx =
01205     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01206   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01207   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01208   ctl->csi_ny =
01209     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01210
01211   /* Output of ensemble data... */
01212   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01213
01214   /* Output of grid data... */
01215   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01216            ctl->grid_basename);
01217   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01218   ctl->grid_dt_out =
01219     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01220   ctl->grid_sparse =
01221     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01222   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01223   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01224   ctl->grid_nz =
01225     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01226   ctl->grid_lon0 =
01227     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01228   ctl->grid_lon1 =
01229     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01230   ctl->grid_nx =
01231     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
```

```
01232   ctl->grid_lat0 =
01233     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01234   ctl->grid_lat1 =
01235     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01236   ctl->grid_ny =
01237     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01238
01239   /* Output of profile data... */
01240   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01241            ctl->prof_basename);
01242   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
01243   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01244   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01245   ctl->prof_nz =
01246     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01247   ctl->prof_lon0 =
01248     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01249   ctl->prof_lon1 =
01250     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01251   ctl->prof_nx =
01252     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01253   ctl->prof_lat0 =
01254     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01255   ctl->prof_lat1 =
01256     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01257   ctl->prof_ny =
01258     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01259
01260   /* Output of station data... */
01261   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01262            ctl->stat_basename);
01263   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01264   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01265   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01266 }
01267
01268 /*****************************************************************************/
01269
01270 void read_met(
01271   ctl_t * ctl,
01272   char *filename,
01273   met_t * met) {
01274
01275   char cmd[2 * LEN], levname[LEN], tstr[10];
01276
01277   static float help[EX * EY];
01278
01279   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01280
01281   size_t np, nx, ny;
01282
01283   /* Write info... */
01284   printf("Read meteorological data: %s\n", filename);
01285
01286   /* Get time from filename... */
01287   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01288   year = atoi(tstr);
01289   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01290   mon = atoi(tstr);
01291   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01292   day = atoi(tstr);
01293   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01294   hour = atoi(tstr);
01295   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01296
01297   /* Open netCDF file... */
01298   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01299
01300     /* Try to stage meteo file... */
01301     STOP_TIMER(TIMER_INPUT);
01302     START_TIMER(TIMER_STAGE);
01303     if (ctl->met_stage[0] != '-') {
01304       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01305               year, mon, day, hour, filename);
01306       if (system(cmd) != 0)
01307         ERRMSG("Error while staging meteo data!");
01308     }
01309     STOP_TIMER(TIMER_STAGE);
01310     START_TIMER(TIMER_INPUT);
01311
01312     /* Try to open again... */
01313     NC(nc_open(filename, NC_NOWRITE, &ncid));
01314   }
01315
01316   /* Get dimensions... */
01317   NC(nc_inq_dimid(ncid, "lon", &dimid));
```

```
01318   NC(nc_inq_dimlen(ncid, dimid, &nx));
01319   if (nx < 2 || nx > EX)
01320     ERRMSG("Number of longitudes out of range!");
01321
01322   NC(nc_inq_dimid(ncid, "lat", &dimid));
01323   NC(nc_inq_dimlen(ncid, dimid, &ny));
01324   if (ny < 2 || ny > EY)
01325     ERRMSG("Number of latitudes out of range!");
01326
01327   sprintf(levname, "lev");
01328   NC(nc_inq_dimid(ncid, levname, &dimid));
01329   NC(nc_inq_dimlen(ncid, dimid, &np));
01330   if (np == 1) {
01331     sprintf(levname, "lev_2");
01332     NC(nc_inq_dimid(ncid, levname, &dimid));
01333     NC(nc_inq_dimlen(ncid, dimid, &np));
01334   }
01335   if (np < 2 || np > EP)
01336     ERRMSG("Number of levels out of range!");
01337
01338   /* Store dimensions... */
01339   met->np = (int) np;
01340   met->nx = (int) nx;
01341   met->ny = (int) ny;
01342
01343   /* Get horizontal grid... */
01344   NC(nc_inq_varid(ncid, "lon", &varid));
01345   NC(nc_get_var_double(ncid, varid, met->lon));
01346   NC(nc_inq_varid(ncid, "lat", &varid));
01347   NC(nc_get_var_double(ncid, varid, met->lat));
01348
01349   /* Read meteorological data... */
01350   read_met_help(ncid, "t", "T", met, met->t, 1.0);
01351   read_met_help(ncid, "u", "U", met, met->u, 1.0);
01352   read_met_help(ncid, "v", "V", met, met->v, 1.0);
01353   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01354   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
01355   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
01356
01357   /* Meteo data on pressure levels... */
01358   if (ctl->met_np <= 0) {
01359
01360     /* Read pressure levels from file... */
01361     NC(nc_inq_varid(ncid, levname, &varid));
01362     NC(nc_get_var_double(ncid, varid, met->p));
01363     for (ip = 0; ip < met->np; ip++)
01364       met->p[ip] /= 100.;
01365
01366     /* Extrapolate data for lower boundary... */
01367     read_met_extrapolate(met);
01368   }
01369
01370   /* Meteo data on model levels... */
01371   else {
01372
01373     /* Read pressure data from file... */
01374     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01375
01376     /* Interpolate from model levels to pressure levels... */
01377     read_met_ml2pl(ctl, met, met->t);
01378     read_met_ml2pl(ctl, met, met->u);
01379     read_met_ml2pl(ctl, met, met->v);
01380     read_met_ml2pl(ctl, met, met->w);
01381     read_met_ml2pl(ctl, met, met->h2o);
01382     read_met_ml2pl(ctl, met, met->o3);
01383
01384     /* Set pressure levels... */
01385     met->np = ctl->met_np;
01386     for (ip = 0; ip < met->np; ip++)
01387       met->p[ip] = ctl->met_p[ip];
01388   }
01389
01390   /* Check ordering of pressure levels... */
01391   for (ip = 1; ip < met->np; ip++)
01392     if (met->p[ip - 1] < met->p[ip])
01393       ERRMSG("Pressure levels must be descending!");
01394
01395   /* Read surface pressure... */
01396   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01397       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01398     NC(nc_get_var_float(ncid, varid, help));
01399     for (iy = 0; iy < met->ny; iy++)
01400       for (ix = 0; ix < met->nx; ix++)
01401         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01402   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01403              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01404     NC(nc_get_var_float(ncid, varid, help));
```

```
01405     for (iy = 0; iy < met->ny; iy++)
01406       for (ix = 0; ix < met->nx; ix++)
01407         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01408   } else
01409     for (ix = 0; ix < met->nx; ix++)
01410       for (iy = 0; iy < met->ny; iy++)
01411         met->ps[ix][iy] = met->p[0];
01412
01413   /* Create periodic boundary conditions... */
01414   read_met_periodic(met);
01415
01416   /* Calculate geopotential heights... */
01417   read_met_geopot(ctl, met);
01418
01419   /* Calculate tropopause pressure... */
01420   read_met_tropo(ctl, met);
01421
01422   /* Downsampling... */
01423   read_met_sample(ctl, met);
01424
01425   /* Close file... */
01426   NC(nc_close(ncid));
01427 }
01428
01429 /*****************************************************************************/
01430
01431 void read_met_extrapolate(
01432   met_t * met) {
01433
01434   int ip, ip0, ix, iy;
01435
01436   /* Loop over columns... */
01437   for (ix = 0; ix < met->nx; ix++)
01438     for (iy = 0; iy < met->ny; iy++) {
01439
01440       /* Find lowest valid data point... */
01441       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01442         if (!gsl_finite(met->t[ix][iy][ip0])
01443             || !gsl_finite(met->u[ix][iy][ip0])
01444             || !gsl_finite(met->v[ix][iy][ip0])
01445             || !gsl_finite(met->w[ix][iy][ip0]))
01446           break;
01447
01448       /* Extrapolate... */
01449       for (ip = ip0; ip >= 0; ip--) {
01450         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01451         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01452         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01453         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01454         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01455         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01456       }
01457     }
01458 }
01459
01460 /*****************************************************************************/
01461
01462 void read_met_geopot(
01463   ctl_t * ctl,
01464   met_t * met) {
01465
01466   static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01467
01468   static int init, topo_nx = -1, topo_ny;
01469
01470   FILE *in;
01471
01472   char line[LEN];
01473
01474   double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01475
01476   float help[EX][EY];
01477
01478   int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01479
01480   /* Initialize geopotential heights... */
01481   for (ix = 0; ix < met->nx; ix++)
01482     for (iy = 0; iy < met->ny; iy++)
01483       for (ip = 0; ip < met->np; ip++)
01484         met->z[ix][iy][ip] = GSL_NAN;
01485
01486   /* Check filename... */
01487   if (ctl->met_geopot[0] == '-')
01488     return;
01489
01490   /* Read surface geopotential... */
01491   if (!init) {
```

```
01492
01493     /* Write info... */
01494     printf("Read surface geopotential: %s\n", ctl->met_geopot);
01495
01496     /* Open file... */
01497     if (!(in = fopen(ctl->met_geopot, "r")))
01498       ERRMSG("Cannot open file!");
01499
01500     /* Read data... */
01501     while (fgets(line, LEN, in))
01502       if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01503         if (rlon != rlon_old) {
01504           if ((++topo_nx) >= EX)
01505             ERRMSG("Too many longitudes!");
01506           topo_ny = 0;
01507         }
01508         rlon_old = rlon;
01509         topo_lon[topo_nx] = rlon;
01510         topo_lat[topo_ny] = rlat;
01511         topo_z[topo_nx][topo_ny] = rz;
01512         if ((++topo_ny) >= EY)
01513           ERRMSG("Too many latitudes!");
01514       }
01515     if ((++topo_nx) >= EX)
01516       ERRMSG("Too many longitudes!");
01517
01518     /* Close file... */
01519     fclose(in);
01520
01521     /* Check grid spacing... */
01522     if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01523         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01524       ERRMSG("Grid spacing does not match!");
01525
01526     /* Set init flag... */
01527     init = 1;
01528   }
01529
01530   /* Apply hydrostatic equation to calculate geopotential heights... */
01531   for (ix = 0; ix < met->nx; ix++)
01532     for (iy = 0; iy < met->ny; iy++) {
01533
01534       /* Get surface height... */
01535       lon = met->lon[ix];
01536       if (lon < topo_lon[0])
01537         lon += 360;
01538       else if (lon > topo_lon[topo_nx - 1])
01539         lon -= 360;
01540       lat = met->lat[iy];
01541       tx = locate(topo_lon, topo_nx, lon);
01542       ty = locate(topo_lat, topo_ny, lat);
01543       z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01544                topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01545       z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01546                topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01547       z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01548
01549       /* Find surface pressure level... */
01550       ip0 = locate(met->p, met->np, met->ps[ix][iy]);
01551
01552       /* Get surface temperature... */
01553       ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
01554                met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
01555
01556       /* Upper part of profile... */
01557       met->z[ix][iy][ip0 + 1]
01558         = (float) (z0 + 8.31441 / 28.9647 / G0
01559                    * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01560                    * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01561       for (ip = ip0 + 2; ip < met->np; ip++)
01562         met->z[ix][iy][ip]
01563           = (float) (met->z[ix][iy][ip - 1] + 8.31441 / 28.9647 / G0
01564                      * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01565                      * log(met->p[ip - 1] / met->p[ip]));
01566     }
01567
01568   /* Smooth fields... */
01569   for (ip = 0; ip < met->np; ip++) {
01570
01571     /* Median filter... */
01572     for (ix = 0; ix < met->nx; ix++)
01573       for (iy = 0; iy < met->nx; iy++) {
01574         n = 0;
01575         for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
01576           ix3 = ix2;
01577           if (ix3 < 0)
01578             ix3 += met->nx;
```

```
01579           if (ix3 >= met->nx)
01580             ix3 -= met->nx;
01581           for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01582                iy2++)
01583             if (gsl_finite(met->z[ix3][iy2][ip])) {
01584               data[n] = met->z[ix3][iy2][ip];
01585               n++;
01586             }
01587         }
01588         if (n > 0) {
01589           gsl_sort(data, 1, (size_t) n);
01590           help[ix][iy] = (float)
01591             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01592         } else
01593           help[ix][iy] = GSL_NAN;
01594       }
01595
01596     /* Copy data... */
01597     for (ix = 0; ix < met->nx; ix++)
01598       for (iy = 0; iy < met->nx; iy++)
01599         met->z[ix][iy][ip] = help[ix][iy];
01600   }
01601 }
01602
01603 /*****************************************************************************/
01604
01605 void read_met_help(
01606   int ncid,
01607   char *varname,
01608   char *varname2,
01609   met_t * met,
01610   float dest[EX][EY][EP],
01611   float scl) {
01612
01613   static float help[EX * EY * EP];
01614
01615   int ip, ix, iy, n = 0, varid;
01616
01617   /* Check if variable exists... */
01618   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01619     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01620       return;
01621
01622   /* Read data... */
01623   NC(nc_get_var_float(ncid, varid, help));
01624
01625   /* Copy and check data... */
01626   for (ip = 0; ip < met->np; ip++)
01627     for (iy = 0; iy < met->ny; iy++)
01628       for (ix = 0; ix < met->nx; ix++) {
01629         dest[ix][iy][ip] = scl * help[n++];
01630         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
01631           dest[ix][iy][ip] = GSL_NAN;
01632       }
01633 }
01634
01635 /*****************************************************************************/
01636
01637 void read_met_ml2pl(
01638   ctl_t * ctl,
01639   met_t * met,
01640   float var[EX][EY][EP]) {
01641
01642   double aux[EP], p[EP], pt;
01643
01644   int ip, ip2, ix, iy;
01645
01646   /* Loop over columns... */
01647   for (ix = 0; ix < met->nx; ix++)
01648     for (iy = 0; iy < met->ny; iy++) {
01649
01650       /* Copy pressure profile... */
01651       for (ip = 0; ip < met->np; ip++)
01652         p[ip] = met->pl[ix][iy][ip];
01653
01654       /* Interpolate... */
01655       for (ip = 0; ip < ctl->met_np; ip++) {
01656         pt = ctl->met_p[ip];
01657         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01658           pt = p[0];
01659         else if ((pt > p[met->np - 1] && p[1] > p[0])
01660                  || (pt < p[met->np - 1] && p[1] < p[0]))
01661           pt = p[met->np - 1];
01662         ip2 = locate(p, met->np, pt);
01663         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01664                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01665       }
```

```
01666
01667       /* Copy data... */
01668       for (ip = 0; ip < ctl->met_np; ip++)
01669         var[ix][iy][ip] = (float) aux[ip];
01670     }
01671 }
01672
01673 /*****************************************************************************/
01674
01675 void read_met_periodic(
01676   met_t * met) {
01677
01678   int ip, iy;
01679
01680   /* Check longitudes... */
01681   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01682              + met->lon[1] - met->lon[0] - 360) < 0.01))
01683     return;
01684
01685   /* Increase longitude counter... */
01686   if ((++met->nx) > EX)
01687     ERRMSG("Cannot create periodic boundary conditions!");
01688
01689   /* Set longitude... */
01690   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01691
01692   /* Loop over latitudes and pressure levels... */
01693   for (iy = 0; iy < met->ny; iy++)
01694     for (ip = 0; ip < met->np; ip++) {
01695       met->ps[met->nx - 1][iy] = met->ps[0][iy];
01696       met->pt[met->nx - 1][iy] = met->pt[0][iy];
01697       met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01698       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01699       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01700       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01701       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01702       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01703       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01704     }
01705 }
01706
01707 /*****************************************************************************/
01708
01709 void read_met_sample(
01710   ctl_t * ctl,
01711   met_t * met) {
01712
01713   met_t *help;
01714
01715   float w, wsum;
01716
01717   int ip, ip2, ix, ix2, iy, iy2;
01718
01719   /* Check parameters... */
01720   if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1)
01721     return;
01722
01723   /* Allocate... */
01724   ALLOC(help, met_t, 1);
01725
01726   /* Copy data... */
01727   help->nx = met->nx;
01728   help->ny = met->ny;
01729   help->np = met->np;
01730   memcpy(help->lon, met->lon, sizeof(met->lon));
01731   memcpy(help->lat, met->lat, sizeof(met->lat));
01732   memcpy(help->p, met->p, sizeof(met->p));
01733
01734   /* Smoothing... */
01735   for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01736     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01737       for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01738         help->ps[ix][iy] = 0;
01739         help->pt[ix][iy] = 0;
01740         help->z[ix][iy][ip] = 0;
01741         help->t[ix][iy][ip] = 0;
01742         help->u[ix][iy][ip] = 0;
01743         help->v[ix][iy][ip] = 0;
01744         help->w[ix][iy][ip] = 0;
01745         help->h2o[ix][iy][ip] = 0;
01746         help->o3[ix][iy][ip] = 0;
01747         wsum = 0;
01748         for (ix2 = GSL_MAX(ix - ctl->met_dx + 1, 0);
01749              ix2 <= GSL_MIN(ix + ctl->met_dx - 1, met->nx - 1); ix2++)
01750           for (iy2 = GSL_MAX(iy - ctl->met_dy + 1, 0);
01751                iy2 <= GSL_MIN(iy + ctl->met_dy - 1, met->ny - 1); iy2++)
```

```
01752             for (ip2 = GSL_MAX(ip - ctl->met_dp + 1, 0);
01753                  ip2 <= GSL_MIN(ip + ctl->met_dp - 1, met->np - 1); ip2++) {
01754               w = (float) (1.0 - fabs(ix - ix2) / ctl->met_dx)
01755                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_dy)
01756                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_dp);
01757               help->ps[ix][iy] += w * met->ps[ix2][iy2];
01758               help->pt[ix][iy] += w * met->pt[ix2][iy2];
01759               help->z[ix][iy][ip] += w * met->z[ix2][iy2][ip2];
01760               help->t[ix][iy][ip] += w * met->t[ix2][iy2][ip2];
01761               help->u[ix][iy][ip] += w * met->u[ix2][iy2][ip2];
01762               help->v[ix][iy][ip] += w * met->v[ix2][iy2][ip2];
01763               help->w[ix][iy][ip] += w * met->w[ix2][iy2][ip2];
01764               help->h2o[ix][iy][ip] += w * met->h2o[ix2][iy2][ip2];
01765               help->o3[ix][iy][ip] += w * met->o3[ix2][iy2][ip2];
01766               wsum += w;
01767             }
01768         help->ps[ix][iy] /= wsum;
01769         help->pt[ix][iy] /= wsum;
01770         help->t[ix][iy][ip] /= wsum;
01771         help->z[ix][iy][ip] /= wsum;
01772         help->u[ix][iy][ip] /= wsum;
01773         help->v[ix][iy][ip] /= wsum;
01774         help->w[ix][iy][ip] /= wsum;
01775         help->h2o[ix][iy][ip] /= wsum;
01776         help->o3[ix][iy][ip] /= wsum;
01777       }
01778     }
01779   }
01780
01781   /* Downsampling... */
01782   met->nx = 0;
01783   for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01784     met->lon[met->nx] = help->lon[ix];
01785     met->ny = 0;
01786     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01787       met->lat[met->ny] = help->lat[iy];
01788       met->ps[met->nx][met->ny] = help->ps[ix][iy];
01789       met->pt[met->nx][met->ny] = help->pt[ix][iy];
01790       met->np = 0;
01791       for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01792         met->p[met->np] = help->p[ip];
01793         met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01794         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01795         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01796         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01797         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01798         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01799         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01800         met->np++;
01801       }
01802       met->ny++;
01803     }
01804     met->nx++;
01805   }
01806
01807   /* Free... */
01808   free(help);
01809 }
01810
01811 /*****************************************************************************/
01812
01813 void read_met_tropo(
01814   ctl_t * ctl,
01815   met_t * met) {
01816
01817   gsl_interp_accel *acc;
01818
01819   gsl_spline *spline;
01820
01821   double tt[400], tt2[400], tz[400], tz2[400];
01822
01823   int found, ix, iy, iz, iz2;
01824
01825   /* Allocate... */
01826   acc = gsl_interp_accel_alloc();
01827   spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01828
01829   /* Do not calculate tropopause... */
01830   if (ctl->met_tropo == 0)
01831     for (ix = 0; ix < met->nx; ix++)
01832       for (iy = 0; iy < met->ny; iy++)
01833         met->pt[ix][iy] = GSL_NAN;
01834
01835   /* Use tropopause climatology... */
01836   else if (ctl->met_tropo == 1)
01837     for (ix = 0; ix < met->nx; ix++)
01838       for (iy = 0; iy < met->ny; iy++)
```

```
01839         met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01840
01841     /* Use cold point... */
01842     else if (ctl->met_tropo == 2) {
01843
01844       /* Loop over grid points... */
01845       for (ix = 0; ix < met->nx; ix++)
01846         for (iy = 0; iy < met->ny; iy++) {
01847
01848           /* Get temperature profile... */
01849           for (iz = 0; iz < met->np; iz++) {
01850             tz[iz] = Z(met->p[iz]);
01851             tt[iz] = met->t[ix][iy][iz];
01852           }
01853
01854           /* Interpolate temperature profile... */
01855           gsl_spline_init(spline, tz, tt, (size_t) met->np);
01856           for (iz = 0; iz <= 170; iz++) {
01857             tz2[iz] = 4.5 + 0.1 * iz;
01858             tt2[iz] = gsl_spline_eval(spline, tz2[iz], acc);
01859           }
01860
01861           /* Find minimum... */
01862           iz = (int) gsl_stats_min_index(tt2, 1, 171);
01863           if (iz <= 0 || iz >= 170)
01864             met->pt[ix][iy] = GSL_NAN;
01865           else
01866             met->pt[ix][iy] = P(tz2[iz]);
01867         }
01868     }
01869
01870     /* Use WMO definition... */
01871     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
01872
01873       /* Loop over grid points... */
01874       for (ix = 0; ix < met->nx; ix++)
01875         for (iy = 0; iy < met->ny; iy++) {
01876
01877           /* Get temperature profile... */
01878           for (iz = 0; iz < met->np; iz++) {
01879             tz[iz] = Z(met->p[iz]);
01880             tt[iz] = met->t[ix][iy][iz];
01881           }
01882
01883           /* Interpolate temperature profile... */
01884           gsl_spline_init(spline, tz, tt, (size_t) met->np);
01885           for (iz = 0; iz <= 160; iz++) {
01886             tz2[iz] = 4.5 + 0.1 * iz;
01887             tt2[iz] = gsl_spline_eval(spline, tz2[iz], acc);
01888           }
01889
01890           /* Find 1st tropopause... */
01891           met->pt[ix][iy] = GSL_NAN;
01892           for (iz = 0; iz <= 140; iz++) {
01893             found = 1;
01894             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
01895               if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01896                   / log(P(tz2[iz2]) / P(tz2[iz])) > 2.0) {
01897                 found = 0;
01898                 break;
01899               }
01900             if (found) {
01901               if (iz > 0 && iz < 140)
01902                 met->pt[ix][iy] = P(tz2[iz]);
01903               break;
01904             }
01905           }
01906
01907           /* Find 2nd tropopause... */
01908           if (ctl->met_tropo == 4) {
01909             met->pt[ix][iy] = GSL_NAN;
01910             for (; iz <= 140; iz++) {
01911               found = 1;
01912               for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
01913                 if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01914                     / log(P(tz2[iz2]) / P(tz2[iz])) < 3.0) {
01915                   found = 0;
01916                   break;
01917                 }
01918               if (found)
01919                 break;
01920             }
01921             for (; iz <= 140; iz++) {
01922               found = 1;
01923               for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
01924                 if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01925                     / log(P(tz2[iz2]) / P(tz2[iz])) > 2.0) {
```

```
01926                 found = 0;
01927                 break;
01928               }
01929             if (found) {
01930               if (iz > 0 && iz < 140)
01931                 met->pt[ix][iy] = P(tz2[iz]);
01932               break;
01933             }
01934           }
01935         }
01936       }
01937   }
01938
01939   else
01940     ERRMSG("Cannot calculate tropopause!");
01941
01942   /* Free... */
01943   gsl_spline_free(spline);
01944   gsl_interp_accel_free(acc);
01945 }
01946
01947 /*****************************************************************************/
01948
01949 double scan_ctl(
01950   const char *filename,
01951   int argc,
01952   char *argv[],
01953   const char *varname,
01954   int arridx,
01955   const char *defvalue,
01956   char *value) {
01957
01958   FILE *in = NULL;
01959
01960   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01961     msg[LEN], rvarname[LEN], rval[LEN];
01962
01963   int contain = 0, i;
01964
01965   /* Open file... */
01966   if (filename[strlen(filename) - 1] != '-')
01967     if (!(in = fopen(filename, "r")))
01968       ERRMSG("Cannot open file!");
01969
01970   /* Set full variable name... */
01971   if (arridx >= 0) {
01972     sprintf(fullname1, "%s[%d]", varname, arridx);
01973     sprintf(fullname2, "%s[*]", varname);
01974   } else {
01975     sprintf(fullname1, "%s", varname);
01976     sprintf(fullname2, "%s", varname);
01977   }
01978
01979   /* Read data... */
01980   if (in != NULL)
01981     while (fgets(line, LEN, in))
01982       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01983         if (strcasecmp(rvarname, fullname1) == 0 ||
01984             strcasecmp(rvarname, fullname2) == 0) {
01985           contain = 1;
01986           break;
01987         }
01988   for (i = 1; i < argc - 1; i++)
01989     if (strcasecmp(argv[i], fullname1) == 0 ||
01990         strcasecmp(argv[i], fullname2) == 0) {
01991       sprintf(rval, "%s", argv[i + 1]);
01992       contain = 1;
01993       break;
01994     }
01995
01996   /* Close file... */
01997   if (in != NULL)
01998     fclose(in);
01999
02000   /* Check for missing variables... */
02001   if (!contain) {
02002     if (strlen(defvalue) > 0)
02003       sprintf(rval, "%s", defvalue);
02004     else {
02005       sprintf(msg, "Missing variable %s!\n", fullname1);
02006       ERRMSG(msg);
02007     }
02008   }
02009
02010   /* Write info... */
02011   printf("%s = %s\n", fullname1, rval);
02012
```

```
02013   /* Return values... */
02014   if (value != NULL)
02015     sprintf(value, "%s", rval);
02016   return atof(rval);
02017 }
02018
02019 /*****************************************************************************/
02020
02021 void time2jsec(
02022   int year,
02023   int mon,
02024   int day,
02025   int hour,
02026   int min,
02027   int sec,
02028   double remain,
02029   double *jsec) {
02030
02031   struct tm t0, t1;
02032
02033   t0.tm_year = 100;
02034   t0.tm_mon = 0;
02035   t0.tm_mday = 1;
02036   t0.tm_hour = 0;
02037   t0.tm_min = 0;
02038   t0.tm_sec = 0;
02039
02040   t1.tm_year = year - 1900;
02041   t1.tm_mon = mon - 1;
02042   t1.tm_mday = day;
02043   t1.tm_hour = hour;
02044   t1.tm_min = min;
02045   t1.tm_sec = sec;
02046
02047   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02048 }
02049
02050 /*****************************************************************************/
02051
02052 void timer(
02053   const char *name,
02054   int id,
02055   int mode) {
02056
02057   static double starttime[NTIMER], runtime[NTIMER];
02058
02059   /* Check id... */
02060   if (id < 0 || id >= NTIMER)
02061     ERRMSG("Too many timers!");
02062
02063   /* Start timer... */
02064   if (mode == 1) {
02065     if (starttime[id] <= 0)
02066       starttime[id] = omp_get_wtime();
02067     else
02068       ERRMSG("Timer already started!");
02069   }
02070
02071   /* Stop timer... */
02072   else if (mode == 2) {
02073     if (starttime[id] > 0) {
02074       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02075       starttime[id] = -1;
02076     }
02077   }
02078
02079   /* Print timer... */
02080   else if (mode == 3)
02081     printf("%s = %.3f s\n", name, runtime[id]);
02082 }
02083
02084 /*****************************************************************************/
02085
02086 void write_atm(
02087   const char *filename,
02088   ctl_t * ctl,
02089   atm_t * atm,
02090   double t) {
02091
02092   FILE *in, *out;
02093
02094   char line[LEN];
02095
02096   double r, t0, t1;
02097
02098   int ip, iq, year, mon, day, hour, min, sec;
02099
```

```
02100   /* Set time interval for output... */
02101   t0 = t - 0.5 * ctl->dt_mod;
02102   t1 = t + 0.5 * ctl->dt_mod;
02103
02104   /* Write info... */
02105   printf("Write atmospheric data: %s\n", filename);
02106
02107   /* Write ASCII data... */
02108   if (ctl->atm_type == 0) {
02109
02110     /* Check if gnuplot output is requested... */
02111     if (ctl->atm_gpfile[0] != '-') {
02112
02113       /* Create gnuplot pipe... */
02114       if (!(out = popen("gnuplot", "w")))
02115         ERRMSG("Cannot create pipe to gnuplot!");
02116
02117       /* Set plot filename... */
02118       fprintf(out, "set out \"%s.png\"\n", filename);
02119
02120       /* Set time string... */
02121       jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02122       fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02123               year, mon, day, hour, min);
02124
02125       /* Dump gnuplot file to pipe... */
02126       if (!(in = fopen(ctl->atm_gpfile, "r")))
02127         ERRMSG("Cannot open file!");
02128       while (fgets(line, LEN, in))
02129         fprintf(out, "%s", line);
02130       fclose(in);
02131     }
02132
02133     else {
02134
02135       /* Create file... */
02136       if (!(out = fopen(filename, "w")))
02137         ERRMSG("Cannot create file!");
02138     }
02139
02140     /* Write header... */
02141     fprintf(out,
02142             "# $1 = time [s]\n"
02143             "# $2 = altitude [km]\n"
02144             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02145     for (iq = 0; iq < ctl->nq; iq++)
02146       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02147               ctl->qnt_unit[iq]);
02148     fprintf(out, "\n");
02149
02150     /* Write data... */
02151     for (ip = 0; ip < atm->np; ip++) {
02152
02153       /* Check time... */
02154       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02155         continue;
02156
02157       /* Write output... */
02158       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02159               atm->lon[ip], atm->lat[ip]);
02160       for (iq = 0; iq < ctl->nq; iq++) {
02161         fprintf(out, " ");
02162         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02163       }
02164       fprintf(out, "\n");
02165     }
02166
02167     /* Close file... */
02168     fclose(out);
02169   }
02170
02171   /* Write binary data... */
02172   else if (ctl->atm_type == 1) {
02173
02174     /* Create file... */
02175     if (!(out = fopen(filename, "w")))
02176       ERRMSG("Cannot create file!");
02177
02178     /* Write data... */
02179     FWRITE(&atm->np, int,
02180            1,
02181            out);
02182     FWRITE(atm->time, double,
02183              (size_t) atm->np,
02184            out);
02185     FWRITE(atm->p, double,
02186              (size_t) atm->np,
```

```
02187             out);
02188     FWRITE(atm->lon, double,
02189               (size_t) atm->np,
02190             out);
02191     FWRITE(atm->lat, double,
02192               (size_t) atm->np,
02193             out);
02194     for (iq = 0; iq < ctl->nq; iq++)
02195       FWRITE(atm->q[iq], double,
02196                 (size_t) atm->np,
02197               out);
02198
02199     /* Close file... */
02200     fclose(out);
02201   }
02202
02203   /* Error... */
02204   else
02205     ERRMSG("Atmospheric data type not supported!");
02206 }
02207
02208 /*****************************************************************************/
02209
02210 void write_csi(
02211   const char *filename,
02212   ctl_t * ctl,
02213   atm_t * atm,
02214   double t) {
02215
02216   static FILE *in, *out;
02217
02218   static char line[LEN];
02219
02220   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02221     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02222
02223   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02224
02225   /* Init... */
02226   if (!init) {
02227     init = 1;
02228
02229     /* Check quantity index for mass... */
02230     if (ctl->qnt_m < 0)
02231       ERRMSG("Need quantity mass to analyze CSI!");
02232
02233     /* Open observation data file... */
02234     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02235     if (!(in = fopen(ctl->csi_obsfile, "r")))
02236       ERRMSG("Cannot open file!");
02237
02238     /* Create new file... */
02239     printf("Write CSI data: %s\n", filename);
02240     if (!(out = fopen(filename, "w")))
02241       ERRMSG("Cannot create file!");
02242
02243     /* Write header... */
02244     fprintf(out,
02245             "# $1 = time [s]\n"
02246             "# $2 = number of hits (cx)\n"
02247             "# $3 = number of misses (cy)\n"
02248             "# $4 = number of false alarms (cz)\n"
02249             "# $5 = number of observations (cx + cy)\n"
02250             "# $6 = number of forecasts (cx + cz)\n"
02251             "# $7 = bias (forecasts/observations) [%%]\n"
02252             "# $8 = probability of detection (POD) [%%]\n"
02253             "# $9 = false alarm rate (FAR) [%%]\n"
02254             "# $10 = critical success index (CSI) [%%]\n\n");
02255   }
02256
02257   /* Set time interval... */
02258   t0 = t - 0.5 * ctl->dt_mod;
02259   t1 = t + 0.5 * ctl->dt_mod;
02260
02261   /* Initialize grid cells... */
02262   for (ix = 0; ix < ctl->csi_nx; ix++)
02263     for (iy = 0; iy < ctl->csi_ny; iy++)
02264       for (iz = 0; iz < ctl->csi_nz; iz++)
02265         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02266
02267   /* Read data... */
02268   while (fgets(line, LEN, in)) {
02269
02270     /* Read data... */
02271     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02272         5)
02273       continue;
```

```
02274
02275     /* Check time... */
02276     if (rt < t0)
02277       continue;
02278     if (rt > t1)
02279       break;
02280
02281     /* Calculate indices... */
02282     ix = (int) ((rlon - ctl->csi_lon0)
02283                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02284     iy = (int) ((rlat - ctl->csi_lat0)
02285                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02286     iz = (int) ((rz - ctl->csi_z0)
02287                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02288
02289     /* Check indices... */
02290     if (ix < 0 || ix >= ctl->csi_nx ||
02291         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02292       continue;
02293
02294     /* Get mean observation index... */
02295     obsmean[ix][iy][iz] += robs;
02296     obscount[ix][iy][iz]++;
02297   }
02298
02299   /* Analyze model data... */
02300   for (ip = 0; ip < atm->np; ip++) {
02301
02302     /* Check time... */
02303     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02304       continue;
02305
02306     /* Get indices... */
02307     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02308                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02309     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02310                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02311     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02312                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02313
02314     /* Check indices... */
02315     if (ix < 0 || ix >= ctl->csi_nx ||
02316         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02317       continue;
02318
02319     /* Get total mass in grid cell... */
02320     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02321   }
02322
02323   /* Analyze all grid cells... */
02324   for (ix = 0; ix < ctl->csi_nx; ix++)
02325     for (iy = 0; iy < ctl->csi_ny; iy++)
02326       for (iz = 0; iz < ctl->csi_nz; iz++) {
02327
02328         /* Calculate mean observation index... */
02329         if (obscount[ix][iy][iz] > 0)
02330           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02331
02332         /* Calculate column density... */
02333         if (modmean[ix][iy][iz] > 0) {
02334           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02335           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02336           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02337           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02338             * cos(lat * M_PI / 180.);
02339           modmean[ix][iy][iz] /= (1e6 * area);
02340         }
02341
02342         /* Calculate CSI... */
02343         if (obscount[ix][iy][iz] > 0) {
02344           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02345               modmean[ix][iy][iz] >= ctl->csi_modmin)
02346             cx++;
02347           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02348                    modmean[ix][iy][iz] < ctl->csi_modmin)
02349             cy++;
02350           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02351                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02352             cz++;
02353         }
02354       }
02355
02356   /* Write output... */
02357   if (fmod(t, ctl->csi_dt_out) == 0) {
02358
02359     /* Write... */
02360     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
```

```
02361             t, cx, cy, cz, cx + cy, cx + cz,
02362             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02363             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02364             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02365             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02366
02367     /* Set counters to zero... */
02368     cx = cy = cz = 0;
02369   }
02370
02371   /* Close file... */
02372   if (t == ctl->t_stop)
02373     fclose(out);
02374 }
02375
02376 /*****************************************************************************/
02377
02378 void write_ens(
02379   const char *filename,
02380   ctl_t * ctl,
02381   atm_t * atm,
02382   double t) {
02383
02384   static FILE *out;
02385
02386   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02387     t0, t1, x[NENS][3], xm[3];
02388
02389   static int init, ip, iq;
02390
02391   static size_t i, n;
02392
02393   /* Init... */
02394   if (!init) {
02395     init = 1;
02396
02397     /* Check quantities... */
02398     if (ctl->qnt_ens < 0)
02399       ERRMSG("Missing ensemble IDs!");
02400
02401     /* Create new file... */
02402     printf("Write ensemble data: %s\n", filename);
02403     if (!(out = fopen(filename, "w")))
02404       ERRMSG("Cannot create file!");
02405
02406     /* Write header... */
02407     fprintf(out,
02408             "# $1 = time [s]\n"
02409             "# $2 = altitude [km]\n"
02410             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02411     for (iq = 0; iq < ctl->nq; iq++)
02412       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02413               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02414     for (iq = 0; iq < ctl->nq; iq++)
02415       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02416               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02417     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02418   }
02419
02420   /* Set time interval... */
02421   t0 = t - 0.5 * ctl->dt_mod;
02422   t1 = t + 0.5 * ctl->dt_mod;
02423
02424   /* Init... */
02425   ens = GSL_NAN;
02426   n = 0;
02427
02428   /* Loop over air parcels... */
02429   for (ip = 0; ip < atm->np; ip++) {
02430
02431     /* Check time... */
02432     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02433       continue;
02434
02435     /* Check ensemble id... */
02436     if (atm->q[ctl->qnt_ens][ip] != ens) {
02437
02438       /* Write results... */
02439       if (n > 0) {
02440
02441         /* Get mean position... */
02442         xm[0] = xm[1] = xm[2] = 0;
02443         for (i = 0; i < n; i++) {
02444           xm[0] += x[i][0] / (double) n;
02445           xm[1] += x[i][1] / (double) n;
02446           xm[2] += x[i][2] / (double) n;
02447         }
```

```
02448         cart2geo(xm, &dummy, &lon, &lat);
02449         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02450                 lat);
02451
02452         /* Get quantity statistics... */
02453         for (iq = 0; iq < ctl->nq; iq++) {
02454           fprintf(out, " ");
02455           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02456         }
02457         for (iq = 0; iq < ctl->nq; iq++) {
02458           fprintf(out, " ");
02459           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02460         }
02461         fprintf(out, " %lu\n", n);
02462       }
02463
02464       /* Init new ensemble... */
02465       ens = atm->q[ctl->qnt_ens][ip];
02466       n = 0;
02467     }
02468
02469     /* Save data... */
02470     p[n] = atm->p[ip];
02471     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02472     for (iq = 0; iq < ctl->nq; iq++)
02473       q[iq][n] = atm->q[iq][ip];
02474     if ((++n) >= NENS)
02475       ERRMSG("Too many data points!");
02476   }
02477
02478   /* Write results... */
02479   if (n > 0) {
02480
02481     /* Get mean position... */
02482     xm[0] = xm[1] = xm[2] = 0;
02483     for (i = 0; i < n; i++) {
02484       xm[0] += x[i][0] / (double) n;
02485       xm[1] += x[i][1] / (double) n;
02486       xm[2] += x[i][2] / (double) n;
02487     }
02488     cart2geo(xm, &dummy, &lon, &lat);
02489     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02490
02491     /* Get quantity statistics... */
02492     for (iq = 0; iq < ctl->nq; iq++) {
02493       fprintf(out, " ");
02494       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02495     }
02496     for (iq = 0; iq < ctl->nq; iq++) {
02497       fprintf(out, " ");
02498       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02499     }
02500     fprintf(out, " %lu\n", n);
02501   }
02502
02503   /* Close file... */
02504   if (t == ctl->t_stop)
02505     fclose(out);
02506 }
02507
02508 /*****************************************************************************/
02509
02510 void write_grid(
02511   const char *filename,
02512   ctl_t * ctl,
02513   met_t * met0,
02514   met_t * met1,
02515   atm_t * atm,
02516   double t) {
02517
02518   FILE *in, *out;
02519
02520   char line[LEN];
02521
02522   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02523     area, rho_air, press, temp, cd, mmr, t0, t1, r;
02524
02525   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02526
02527   /* Check dimensions... */
02528   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02529     ERRMSG("Grid dimensions too large!");
02530
02531   /* Check quantity index for mass... */
02532   if (ctl->qnt_m < 0)
02533     ERRMSG("Need quantity mass to write grid data!");
02534
```

```
02535   /* Set time interval for output... */
02536   t0 = t - 0.5 * ctl->dt_mod;
02537   t1 = t + 0.5 * ctl->dt_mod;
02538
02539   /* Set grid box size... */
02540   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02541   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02542   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02543
02544   /* Initialize grid... */
02545   for (ix = 0; ix < ctl->grid_nx; ix++)
02546     for (iy = 0; iy < ctl->grid_ny; iy++)
02547       for (iz = 0; iz < ctl->grid_nz; iz++)
02548         grid_m[ix][iy][iz] = 0;
02549
02550   /* Average data... */
02551   for (ip = 0; ip < atm->np; ip++)
02552     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02553
02554       /* Get index... */
02555       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02556       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02557       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02558
02559       /* Check indices... */
02560       if (ix < 0 || ix >= ctl->grid_nx ||
02561           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02562         continue;
02563
02564       /* Add mass... */
02565       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02566     }
02567
02568   /* Check if gnuplot output is requested... */
02569   if (ctl->grid_gpfile[0] != '-') {
02570
02571     /* Write info... */
02572     printf("Plot grid data: %s.png\n", filename);
02573
02574     /* Create gnuplot pipe... */
02575     if (!(out = popen("gnuplot", "w")))
02576       ERRMSG("Cannot create pipe to gnuplot!");
02577
02578     /* Set plot filename... */
02579     fprintf(out, "set out \"%s.png\"\n", filename);
02580
02581     /* Set time string... */
02582     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02583     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02584             year, mon, day, hour, min);
02585
02586     /* Dump gnuplot file to pipe... */
02587     if (!(in = fopen(ctl->grid_gpfile, "r")))
02588       ERRMSG("Cannot open file!");
02589     while (fgets(line, LEN, in))
02590       fprintf(out, "%s", line);
02591     fclose(in);
02592   }
02593
02594   else {
02595
02596     /* Write info... */
02597     printf("Write grid data: %s\n", filename);
02598
02599     /* Create file... */
02600     if (!(out = fopen(filename, "w")))
02601       ERRMSG("Cannot create file!");
02602   }
02603
02604   /* Write header... */
02605   fprintf(out,
02606           "# $1 = time [s]\n"
02607           "# $2 = altitude [km]\n"
02608           "# $3 = longitude [deg]\n"
02609           "# $4 = latitude [deg]\n"
02610           "# $5 = surface area [km^2]\n"
02611           "# $6 = layer width [km]\n"
02612           "# $7 = temperature [K]\n"
02613           "# $8 = column density [kg/m^2]\n"
02614           "# $9 = mass mixing ratio [1]\n\n");
02615
02616   /* Write data... */
02617   for (ix = 0; ix < ctl->grid_nx; ix++) {
02618     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02619       fprintf(out, "\n");
02620     for (iy = 0; iy < ctl->grid_ny; iy++) {
02621       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
```

```
02622         fprintf(out, "\n");
02623       for (iz = 0; iz < ctl->grid_nz; iz++)
02624         if (!ctl->grid_sparse
02625             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
02626
02627           /* Set coordinates... */
02628           z = ctl->grid_z0 + dz * (iz + 0.5);
02629           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02630           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02631
02632           /* Get pressure and temperature... */
02633           press = P(z);
02634           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02635                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
02636
02637           /* Calculate surface area... */
02638           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
02639             * cos(lat * M_PI / 180.);
02640
02641           /* Calculate column density... */
02642           cd = grid_m[ix][iy][iz] / (1e6 * area);
02643
02644           /* Calculate mass mixing ratio... */
02645           rho_air = 100. * press / (R0 * temp);
02646           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
02647
02648           /* Write output... */
02649           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
02650                   t, z, lon, lat, area, dz, temp, cd, mmr);
02651         }
02652     }
02653   }
02654
02655   /* Close file... */
02656   fclose(out);
02657 }
02658
02659 /*****************************************************************************/
02660
02661 void write_prof(
02662   const char *filename,
02663   ctl_t * ctl,
02664   met_t * met0,
02665   met_t * met1,
02666   atm_t * atm,
02667   double t) {
02668
02669   static FILE *in, *out;
02670
02671   static char line[LEN];
02672
02673   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
02674     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
02675     press, temp, rho_air, mmr, h2o, o3;
02676
02677   static int init, obscount[GX][GY], ip, ix, iy, iz;
02678
02679   /* Init... */
02680   if (!init) {
02681     init = 1;
02682
02683     /* Check quantity index for mass... */
02684     if (ctl->qnt_m < 0)
02685       ERRMSG("Need quantity mass!");
02686
02687     /* Check dimensions... */
02688     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02689       ERRMSG("Grid dimensions too large!");
02690
02691     /* Open observation data file... */
02692     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02693     if (!(in = fopen(ctl->prof_obsfile, "r")))
02694       ERRMSG("Cannot open file!");
02695
02696     /* Create new file... */
02697     printf("Write profile data: %s\n", filename);
02698     if (!(out = fopen(filename, "w")))
02699       ERRMSG("Cannot create file!");
02700
02701     /* Write header... */
02702     fprintf(out,
02703             "# $1  = time [s]\n"
02704             "# $2  = altitude [km]\n"
02705             "# $3  = longitude [deg]\n"
02706             "# $4  = latitude [deg]\n"
02707             "# $5  = pressure [hPa]\n"
02708             "# $6  = temperature [K]\n"
```

```
02709             "# $7  = mass mixing ratio [1]\n"
02710             "# $8  = H2O volume mixing ratio [1]\n"
02711             "# $9  = O3 volume mixing ratio [1]\n"
02712             "# $10 = mean BT index [K]\n");
02713
02714     /* Set grid box size... */
02715     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
02716     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02717     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02718   }
02719
02720   /* Set time interval... */
02721   t0 = t - 0.5 * ctl->dt_mod;
02722   t1 = t + 0.5 * ctl->dt_mod;
02723
02724   /* Initialize... */
02725   for (ix = 0; ix < ctl->prof_nx; ix++)
02726     for (iy = 0; iy < ctl->prof_ny; iy++) {
02727       obsmean[ix][iy] = 0;
02728       obscount[ix][iy] = 0;
02729       tmean[ix][iy] = 0;
02730       for (iz = 0; iz < ctl->prof_nz; iz++)
02731         mass[ix][iy][iz] = 0;
02732     }
02733
02734   /* Read data... */
02735   while (fgets(line, LEN, in)) {
02736
02737     /* Read data... */
02738     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
02739       continue;
02740
02741     /* Check time... */
02742     if (rt < t0)
02743       continue;
02744     if (rt > t1)
02745       break;
02746
02747     /* Calculate indices... */
02748     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
02749     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02750
02751     /* Check indices... */
02752     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02753       continue;
02754
02755     /* Get mean observation index... */
02756     obsmean[ix][iy] += robs;
02757     tmean[ix][iy] += rt;
02758     obscount[ix][iy]++;
02759   }
02760
02761   /* Analyze model data... */
02762   for (ip = 0; ip < atm->np; ip++) {
02763
02764     /* Check time... */
02765     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02766       continue;
02767
02768     /* Get indices... */
02769     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
02770     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02771     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02772
02773     /* Check indices... */
02774     if (ix < 0 || ix >= ctl->prof_nx ||
02775         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02776       continue;
02777
02778     /* Get total mass in grid cell... */
02779     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02780   }
02781
02782   /* Extract profiles... */
02783   for (ix = 0; ix < ctl->prof_nx; ix++)
02784     for (iy = 0; iy < ctl->prof_ny; iy++)
02785       if (obscount[ix][iy] > 0) {
02786
02787         /* Write output... */
02788         fprintf(out, "\n");
02789
02790         /* Loop over altitudes... */
02791         for (iz = 0; iz < ctl->prof_nz; iz++) {
02792
02793           /* Set coordinates... */
02794           z = ctl->prof_z0 + dz * (iz + 0.5);
02795           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
```

```
02796           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
02797
02798           /* Get meteorological data... */
02799           press = P(z);
02800           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02801                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02802
02803           /* Calculate mass mixing ratio... */
02804           rho_air = 100. * press / (R0 * temp);
02805           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
02806             * cos(lat * M_PI / 180.);
02807           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02808
02809           /* Write output... */
02810           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02811                   tmean[ix][iy] / obscount[ix][iy],
02812                   z, lon, lat, press, temp, mmr, h2o, o3,
02813                   obsmean[ix][iy] / obscount[ix][iy]);
02814         }
02815       }
02816
02817   /* Close file... */
02818   if (t == ctl->t_stop)
02819     fclose(out);
02820 }
02821
02822 /*****************************************************************************/
02823
02824 void write_station(
02825   const char *filename,
02826   ctl_t * ctl,
02827   atm_t * atm,
02828   double t) {
02829
02830   static FILE *out;
02831
02832   static double rmax2, t0, t1, x0[3], x1[3];
02833
02834   static int init, ip, iq;
02835
02836   /* Init... */
02837   if (!init) {
02838     init = 1;
02839
02840     /* Write info... */
02841     printf("Write station data: %s\n", filename);
02842
02843     /* Create new file... */
02844     if (!(out = fopen(filename, "w")))
02845       ERRMSG("Cannot create file!");
02846
02847     /* Write header... */
02848     fprintf(out,
02849             "# $1 = time [s]\n"
02850             "# $2 = altitude [km]\n"
02851             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02852     for (iq = 0; iq < ctl->nq; iq++)
02853       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
02854               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02855     fprintf(out, "\n");
02856
02857     /* Set geolocation and search radius... */
02858     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02859     rmax2 = gsl_pow_2(ctl->stat_r);
02860   }
02861
02862   /* Set time interval for output... */
02863   t0 = t - 0.5 * ctl->dt_mod;
02864   t1 = t + 0.5 * ctl->dt_mod;
02865
02866   /* Loop over air parcels... */
02867   for (ip = 0; ip < atm->np; ip++) {
02868
02869     /* Check time... */
02870     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02871       continue;
02872
02873     /* Check station flag... */
02874     if (ctl->qnt_stat >= 0)
02875       if (atm->q[ctl->qnt_stat][ip])
02876         continue;
02877
02878     /* Get Cartesian coordinates... */
02879     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02880
02881     /* Check horizontal distance... */
02882     if (DIST2(x0, x1) > rmax2)
```

```
02883       continue;
02884
02885     /* Set station flag... */
02886     if (ctl->qnt_stat >= 0)
02887       atm->q[ctl->qnt_stat][ip] = 1;
02888
02889     /* Write data... */
02890     fprintf(out, "%.2f %g %g %g",
02891             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02892     for (iq = 0; iq < ctl->nq; iq++) {
02893       fprintf(out, " ");
02894       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02895     }
02896     fprintf(out, "\n");
02897   }
02898
02899   /* Close file... */
02900   if (t == ctl->t_stop)
02901     fclose(out);
02902 }
```

5.17 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 clim_hno3 (double t, double lat, double p)

  Climatology of HNO3 volume mixing ratios.
- double clim_tropo (double t, double lat)

  Climatology of tropopause pressure.
- 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 *pt, double *z, 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 *pt, double *z, 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_geopot (ctl_t *ctl, met_t *met)

  Calculate geopotential heights.
- 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.
- void read_met_sample (ctl_t *ctl, met_t *met)

  Downsampling of meteorological data.
- void read_met_tropo (ctl_t *ctl, met_t *met)

  Calculate tropopause pressure.
- 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.
- 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.

5.17.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

5.17.2 Function Documentation

5.17.2.1 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 }
```

5.17.2.2 double clim_hno3 (double *t*, double *lat*, double *p*)

Climatology of HNO3 volume mixing ratios.

Definition at line 45 of file libtrac.c.

```
00048            {
00049
00050   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051     9072000.00, 11664000.00, 14342400.00,
00052     16934400.00, 19612800.00, 22291200.00,
00053     24883200.00, 27561600.00, 30153600.00
00054   };
00055
00056   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00057     5, 15, 25, 35, 45, 55, 65, 75, 85
00058   };
00059
00060   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00061     31.6228, 46.4159, 68.1292, 100, 146.78
00062   };
00063
00064   static double hno3[12][18][10] = {
00065     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00066      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00067      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00068      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00069      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00070      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00071      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00072      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
```

```
00073      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00074      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00075      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00076      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00077      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00078      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00079      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00080      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00081      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00082      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00083     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00084      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00085      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00086      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00087      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00088      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00089      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00090      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00091      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00092      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00093      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00094      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00095      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00096      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00097      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00098      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00099      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00100      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00101     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00102      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00103      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00104      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00105      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00106      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00107      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00108      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00109      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00110      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00111      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00112      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00113      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00114      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00115      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00116      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00117      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00118      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00119     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00120      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00121      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00122      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00123      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00124      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00125      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00126      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00127      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00128      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00129      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00130      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00131      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00132      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00133      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00134      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00135      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00136      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00137     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00138      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00139      {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00140      {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00141      {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00142      {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00143      {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00144      {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00145      {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00146      {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00147      {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00148      {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00149      {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00150      {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00151      {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00152      {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00153      {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00154      {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00155     {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00156      {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00157      {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00158      {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00159      {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
```

```
00160       {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00161       {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00162       {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00163       {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00164       {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00165       {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00166       {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00167       {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00168       {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00169       {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00170       {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00171       {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00172       {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00173      {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00174       {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00175       {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00176       {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00177       {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00178       {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00179       {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00180       {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00181       {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00182       {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00183       {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00184       {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00185       {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00186       {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00187       {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00188       {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00189       {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00190       {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00191      {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00192       {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00193       {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00194       {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00195       {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00196       {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00197       {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00198       {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00199       {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00200       {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00201       {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00202       {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00203       {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00204       {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00205       {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00206       {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00207       {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00208       {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00209      {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00210       {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00211       {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00212       {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00213       {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00214       {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00215       {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00216       {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00217       {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00218       {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00219       {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00220       {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00221       {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00222       {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00223       {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00224       {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00225       {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00226       {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00227      {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00228       {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00229       {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00230       {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00231       {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00232       {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00233       {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00234       {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00235       {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00236       {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00237       {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00238       {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00239       {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00240       {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00241       {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00242       {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00243       {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00244       {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00245      {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00246       {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
```

```
00247       {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00248       {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00249       {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00250       {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00251       {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00252       {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00253       {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00254       {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00255       {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00256       {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00257       {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00258       {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00259       {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00260       {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00261       {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00262       {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00263     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00264       {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00265       {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00266       {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00267       {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00268       {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00269       {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00270       {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00271       {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00272       {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00273       {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00274       {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00275       {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00276       {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00277       {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00278       {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00279       {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00280       {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00281   };
00282
00283   double aux00, aux01, aux10, aux11, sec;
00284
00285   int ilat, ip, isec;
00286
00287   /* Get seconds since begin of year... */
00288   sec = fmod(t, 365.25 * 86400.);
00289
00290   /* Get indices... */
00291   ilat = locate(lats, 18, lat);
00292   ip = locate(ps, 10, p);
00293   isec = locate(secs, 12, sec);
00294
00295   /* Interpolate... */
00296   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
00297               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
00298   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
00299               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
00300   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
00301               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
00302   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
00303               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
00304   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
00305   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
00306   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
00307 }
```

Here is the call graph for this function:

5.17.2.3 double clim_tropo (double *t*, double *lat*)

Climatology of tropopause pressure.

Definition at line 311 of file libtrac.c.

```
00313               {
00314
00315   static double doys[12]
00316   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317
00318   static double lats[73]
00319     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00320     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00321     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
```

```
00322     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00323     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00324     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00325     75, 77.5, 80, 82.5, 85, 87.5, 90
00326   };
00327
00328   static double tps[12][73]
00329     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00330          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00331          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00332          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00333          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00334          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00335          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00336          275.3, 275.6, 275.4, 274.1, 273.5},
00337   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00338    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00339    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00340    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00341    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00342    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00343    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00344    287.5, 286.2, 285.8},
00345   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00346    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00347    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00348    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00349    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00350    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00351    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00352    304.3, 304.9, 306, 306.6, 306.2, 306},
00353   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00354    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00355    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00356    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00357    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00358    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00359    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00360    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00361   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00362    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00363    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00364    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00365    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00366    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00367    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00368    325.3, 325.8, 325.8},
00369   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00370    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00371    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00372    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00373    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00374    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00375    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00376    308.5, 312.2, 313.1, 313.3},
00377   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00378    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00379    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00380    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00381    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00382    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00383    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00384    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00385   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00386    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00387    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00388    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00389    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00390    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00391    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00392    278.2, 282.6, 287.4, 290.9, 292.5, 293},
00393   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00394    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00395    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00396    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00397    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00398    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00399    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00400    276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00401   {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00402    215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00403    237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00404    111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00405    106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00406    112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00407    206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00408    279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
```

```
00409    305.1},
00410   {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00411    253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00412    223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00413    108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00414    102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00415    109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00416    241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00417    286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00418   {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00419    284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00420    175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00421    100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00422    100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00423    186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00424    280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00425    281.7, 281.1, 281.2}
00426   };
00427
00428   double doy, p0, p1, pt;
00429
00430   int imon, ilat;
00431
00432   /* Get day of year... */
00433   doy = fmod(t / 86400., 365.25);
00434   while (doy < 0)
00435     doy += 365.25;
00436
00437   /* Get indices... */
00438   imon = locate(doys, 12, doy);
00439   ilat = locate(lats, 73, lat);
00440
00441   /* Get tropopause pressure... */
00442   p0 = LIN(lats[ilat], tps[imon][ilat],
00443            lats[ilat + 1], tps[imon][ilat + 1], lat);
00444   p1 = LIN(lats[ilat], tps[imon + 1][ilat],
00445            lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
00446   pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447
00448   /* Return tropopause pressure... */
00449   return pt;
00450 }
```

Here is the call graph for this function:

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

Convert degrees to horizontal distance.

Definition at line 454 of file libtrac.c.

```
00456              {
00457
00458   return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00459 }
```

5.17.2.5 double deg2dy (double *dlat*)

Convert degrees to horizontal distance.

Definition at line 463 of file libtrac.c.

```
00464               {
00465
00466   return dlat * M_PI * RE / 180.;
00467 }
```

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

Convert pressure to vertical distance.

Definition at line 471 of file libtrac.c.

```
00473             {
00474
00475   return -dp * H0 / p;
00476 }
```

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

Convert horizontal distance to degrees.

Definition at line 480 of file libtrac.c.

```
00482               {
00483
00484   /* Avoid singularity at poles... */
00485   if (lat < -89.999 || lat > 89.999)
00486     return 0;
00487   else
00488     return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00489 }
```

5.17.2.8 double dy2deg (double *dy*)

Convert horizontal distance to degrees.

Definition at line 493 of file libtrac.c.

```
00494              {
00495
00496   return dy * 180. / (M_PI * RE);
00497 }
```

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

Convert vertical distance to pressure.

Definition at line 501 of file libtrac.c.

```
00503             {
00504
00505   return -dz * p / H0;
00506 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 510 of file libtrac.c.

```
00514              {
00515
00516   double radius;
00517
00518   radius = z + RE;
00519   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00520   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00521   x[2] = radius * sin(lat / 180 * M_PI);
00522 }
```

5.17.2.11 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 526 of file libtrac.c.

```
00531                 {
00532
00533   char filename[LEN];
00534
00535   static int init;
00536
00537   /* Init... */
00538   if (!init) {
00539     init = 1;
00540
00541     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00542     read_met(ctl, filename, met0);
00543
00544     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00545     read_met(ctl, filename, met1);
00546   }
00547
00548   /* Read new data for forward trajectories... */
00549   if (t > met1->time && ctl->direction == 1) {
00550     memcpy(met0, met1, sizeof(met_t));
00551     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00552     read_met(ctl, filename, met1);
00553   }
00554
00555   /* Read new data for backward trajectories... */
00556   if (t < met0->time && ctl->direction == -1) {
00557     memcpy(met1, met0, sizeof(met_t));
00558     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00559     read_met(ctl, filename, met0);
00560   }
00561 }
```

Here is the call graph for this function:

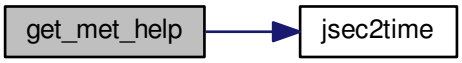
5.17.2.12 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

Get meteorological data for timestep.

Definition at line 565 of file libtrac.c.

```
00570                   {
00571
00572   double t6, r;
00573
00574   int year, mon, day, hour, min, sec;
00575
00576   /* Round time to fixed intervals... */
00577   if (direct == -1)
00578     t6 = floor(t / dt_met) * dt_met;
00579   else
00580     t6 = ceil(t / dt_met) * dt_met;
00581
00582   /* Decode time... */
00583   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00584
00585   /* Set filename... */
00586   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00587 }
```

Here is the call graph for this function:



5.17.2.13 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 591 of file libtrac.c.

```
00597                {
00598
00599   double aux00, aux01, aux10, aux11;
00600
00601   /* Set variables... */
00602   aux00 = array[ix][iy];
00603   aux01 = array[ix][iy + 1];
00604   aux10 = array[ix + 1][iy];
00605   aux11 = array[ix + 1][iy + 1];
00606
00607   /* Interpolate horizontally... */
00608   aux00 = wy * (aux00 - aux01) + aux01;
00609   aux11 = wy * (aux10 - aux11) + aux11;
00610   *var = wx * (aux00 - aux11) + aux11;
00611 }
```

5.17.2.14 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 615 of file libtrac.c.

```
00623                {
00624
00625   double aux00, aux01, aux10, aux11;
00626
00627   /* Interpolate vertically... */
00628   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00629     + array[ix][iy][ip + 1];
00630   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00631     + array[ix][iy + 1][ip + 1];
00632   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00633     + array[ix + 1][iy][ip + 1];
00634   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00635     + array[ix + 1][iy + 1][ip + 1];
00636
00637   /* Interpolate horizontally... */
00638   aux00 = wy * (aux00 - aux01) + aux01;
00639   aux11 = wy * (aux10 - aux11) + aux11;
00640   *var = wx * (aux00 - aux11) + aux11;
00641 }
```

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

Spatial interpolation of meteorological data.

Definition at line 645 of file libtrac.c.

```
00658                {
00659
00660   double wp, wx, wy;
00661
00662   int ip, ix, iy;
00663
00664   /* Check longitude... */
00665   if (met->lon[met->nx - 1] > 180 && lon < 0)
00666     lon += 360;
00667
00668   /* Get indices... */
00669   ip = locate(met->p, met->np, p);
00670   ix = locate(met->lon, met->nx, lon);
00671   iy = locate(met->lat, met->ny, lat);
```

```
00672
00673   /* Get weights... */
00674   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00675   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00676   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00677
00678   /* Interpolate... */
00679   if (ps != NULL)
00680     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00681   if (pt != NULL)
00682     intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00683   if (z != NULL)
00684     intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00685   if (t != NULL)
00686     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00687   if (u != NULL)
00688     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00689   if (v != NULL)
00690     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00691   if (w != NULL)
00692     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00693   if (h2o != NULL)
00694     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00695   if (o3 != NULL)
00696     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00697 }
```

Here is the call graph for this function:

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

Temporal interpolation of meteorological data.

Definition at line 701 of file libtrac.c.

```
00716               {
00717
00718   double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, t0, t1, u0, u1, v0, v1,
00719     w0, w1, wt, z0, z1;
00720
00721   /* Spatial interpolation... */
00722   intpol_met_space(met0, p, lon, lat,
00723                    ps == NULL ? NULL : &ps0,
00724                    pt == NULL ? NULL : &pt0,
00725                    z == NULL ? NULL : &z0,
00726                    t == NULL ? NULL : &t0,
00727                    u == NULL ? NULL : &u0,
00728                    v == NULL ? NULL : &v0,
00729                    w == NULL ? NULL : &w0,
00730                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00731   intpol_met_space(met1, p, lon, lat,
00732                    ps == NULL ? NULL : &ps1,
00733                    pt == NULL ? NULL : &pt1,
00734                    z == NULL ? NULL : &z1,
00735                    t == NULL ? NULL : &t1,
00736                    u == NULL ? NULL : &u1,
00737                    v == NULL ? NULL : &v1,
00738                    w == NULL ? NULL : &w1,
00739                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00740
00741   /* Get weighting factor... */
00742   wt = (met1->time - ts) / (met1->time - met0->time);
00743
00744   /* Interpolate... */
00745   if (ps != NULL)
00746     *ps = wt * (ps0 - ps1) + ps1;
00747   if (pt != NULL)
00748     *pt = wt * (pt0 - pt1) + pt1;
00749   if (z != NULL)
00750     *z = wt * (z0 - z1) + z1;
00751   if (t != NULL)
00752     *t = wt * (t0 - t1) + t1;
00753   if (u != NULL)
00754     *u = wt * (u0 - u1) + u1;
00755   if (v != NULL)
00756     *v = wt * (v0 - v1) + v1;
00757   if (w != NULL)
00758     *w = wt * (w0 - w1) + w1;
```

```
00759   if (h2o != NULL)
00760     *h2o = wt * (h2o0 - h2o1) + h2o1;
00761   if (o3 != NULL)
00762     *o3 = wt * (o30 - o31) + o31;
00763 }
```

Here is the call graph for this function:

5.17.2.17 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 767 of file libtrac.c.

```
00775                   {
00776
00777   struct tm t0, *t1;
00778
00779   time_t jsec0;
00780
00781   t0.tm_year = 100;
00782   t0.tm_mon = 0;
00783   t0.tm_mday = 1;
00784   t0.tm_hour = 0;
00785   t0.tm_min = 0;
00786   t0.tm_sec = 0;
00787
00788   jsec0 = (time_t) jsec + timegm(&t0);
00789   t1 = gmtime(&jsec0);
00790
00791   *year = t1->tm_year + 1900;
00792   *mon = t1->tm_mon + 1;
00793   *day = t1->tm_mday;
00794   *hour = t1->tm_hour;
00795   *min = t1->tm_min;
00796   *sec = t1->tm_sec;
00797   *remain = jsec - floor(jsec);
00798 }
```

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

Find array index.

Definition at line 802 of file libtrac.c.

```
00805             {
00806
00807   int i, ilo, ihi;
00808
00809   ilo = 0;
00810   ihi = n - 1;
00811   i = (ihi + ilo) >> 1;
00812
00813   if (xx[i] < xx[i + 1])
00814     while (ihi > ilo + 1) {
00815       i = (ihi + ilo) >> 1;
00816       if (xx[i] > x)
00817         ihi = i;
00818       else
00819         ilo = i;
00820   } else
00821     while (ihi > ilo + 1) {
00822       i = (ihi + ilo) >> 1;
00823       if (xx[i] <= x)
00824         ihi = i;
00825       else
00826         ilo = i;
00827     }
00828
00829   return ilo;
00830 }
```

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

Read atmospheric data.

Definition at line 834 of file libtrac.c.

```
00837               {
00838
00839   FILE *in;
00840
00841   char line[LEN], *tok;
00842
00843   double t0;
00844
00845   int dimid, ip, iq, ncid, varid;
00846
00847   size_t nparts;
00848
00849   /* Init... */
00850   atm->np = 0;
00851
00852   /* Write info... */
00853   printf("Read atmospheric data: %s\n", filename);
00854
00855   /* Read ASCII data... */
00856   if (ctl->atm_type == 0) {
00857
00858     /* Open file... */
00859     if (!(in = fopen(filename, "r")))
00860       ERRMSG("Cannot open file!");
00861
00862     /* Read line... */
00863     while (fgets(line, LEN, in)) {
00864
00865       /* Read data... */
00866       TOK(line, tok, "%lg", atm->time[atm->np]);
00867       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00868       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00869       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00870       for (iq = 0; iq < ctl->nq; iq++)
00871         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00872
00873       /* Convert altitude to pressure... */
00874       atm->p[atm->np] = P(atm->p[atm->np]);
00875
00876       /* Increment data point counter... */
00877       if ((++atm->np) > NP)
00878         ERRMSG("Too many data points!");
00879     }
00880
00881     /* Close file... */
00882     fclose(in);
00883   }
00884
00885   /* Read binary data... */
00886   else if (ctl->atm_type == 1) {
00887
00888     /* Open file... */
00889     if (!(in = fopen(filename, "r")))
00890       ERRMSG("Cannot open file!");
00891
00892     /* Read data... */
00893     FREAD(&atm->np, int,
00894           1,
00895           in);
00896     FREAD(atm->time, double,
00897             (size_t) atm->np,
00898           in);
00899     FREAD(atm->p, double,
00900             (size_t) atm->np,
00901           in);
00902     FREAD(atm->lon, double,
00903             (size_t) atm->np,
00904           in);
00905     FREAD(atm->lat, double,
00906             (size_t) atm->np,
00907           in);
00908     for (iq = 0; iq < ctl->nq; iq++)
00909       FREAD(atm->q[iq], double,
00910               (size_t) atm->np,
00911             in);
00912
00913     /* Close file... */
```

```
00914     fclose(in);
00915   }
00916
00917   /* Read netCDF data... */
00918   else if (ctl->atm_type == 2) {
00919
00920     /* Open file... */
00921     NC(nc_open(filename, NC_NOWRITE, &ncid));
00922
00923     /* Get dimensions... */
00924     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00925     NC(nc_inq_dimlen(ncid, dimid, &nparts));
00926     atm->np = (int) nparts;
00927     if (atm->np > NP)
00928       ERRMSG("Too many particles!");
00929
00930     /* Get time... */
00931     NC(nc_inq_varid(ncid, "time", &varid));
00932     NC(nc_get_var_double(ncid, varid, &t0));
00933     for (ip = 0; ip < atm->np; ip++)
00934       atm->time[ip] = t0;
00935
00936     /* Read geolocations... */
00937     NC(nc_inq_varid(ncid, "PRESS", &varid));
00938     NC(nc_get_var_double(ncid, varid, atm->p));
00939     NC(nc_inq_varid(ncid, "LON", &varid));
00940     NC(nc_get_var_double(ncid, varid, atm->lon));
00941     NC(nc_inq_varid(ncid, "LAT", &varid));
00942     NC(nc_get_var_double(ncid, varid, atm->lat));
00943
00944     /* Read variables... */
00945     if (ctl->qnt_p >= 0)
00946       if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
00947         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
00948     if (ctl->qnt_t >= 0)
00949       if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
00950         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
00951     if (ctl->qnt_u >= 0)
00952       if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
00953         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
00954     if (ctl->qnt_v >= 0)
00955       if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
00956         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
00957     if (ctl->qnt_w >= 0)
00958       if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
00959         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
00960     if (ctl->qnt_h2o >= 0)
00961       if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
00962         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
00963     if (ctl->qnt_o3 >= 0)
00964       if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
00965         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
00966     if (ctl->qnt_theta >= 0)
00967       if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
00968         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
00969     if (ctl->qnt_pv >= 0)
00970       if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
00971         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
00972
00973     /* Check data... */
00974     for (ip = 0; ip < atm->np; ip++)
00975       if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
00976           || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
00977           || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
00978           || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
00979           || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
00980         atm->time[ip] = GSL_NAN;
00981         atm->p[ip] = GSL_NAN;
00982         atm->lon[ip] = GSL_NAN;
00983         atm->lat[ip] = GSL_NAN;
00984         for (iq = 0; iq < ctl->nq; iq++)
00985           atm->q[iq][ip] = GSL_NAN;
00986       } else {
00987         if (ctl->qnt_h2o >= 0)
00988           atm->q[ctl->qnt_h2o][ip] *= 1.608;
00989         if (atm->lon[ip] > 180)
00990           atm->lon[ip] -= 360;
00991       }
00992
00993     /* Close file... */
00994     NC(nc_close(ncid));
00995   }
00996
00997   /* Error... */
00998   else
00999     ERRMSG("Atmospheric data type not supported!");
01000
```

```
01001   /* Check number of points... */
01002   if (atm->np < 1)
01003     ERRMSG("Can not read any data!");
01004 }
```

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

Read control parameters.

Definition at line 1008 of file libtrac.c.

```
01012               {
01013
01014   int ip, iq;
01015
01016   /* Write info... */
01017   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01018          "(executable: %s | compiled: %s, %s)\n\n",
01019          argv[0], __DATE__, __TIME__);
01020
01021   /* Initialize quantity indices... */
01022   ctl->qnt_ens = -1;
01023   ctl->qnt_m = -1;
01024   ctl->qnt_r = -1;
01025   ctl->qnt_rho = -1;
01026   ctl->qnt_ps = -1;
01027   ctl->qnt_pt = -1;
01028   ctl->qnt_z = -1;
01029   ctl->qnt_p = -1;
01030   ctl->qnt_t = -1;
01031   ctl->qnt_u = -1;
01032   ctl->qnt_v = -1;
01033   ctl->qnt_w = -1;
01034   ctl->qnt_h2o = -1;
01035   ctl->qnt_o3 = -1;
01036   ctl->qnt_theta = -1;
01037   ctl->qnt_pv = -1;
01038   ctl->qnt_tice = -1;
01039   ctl->qnt_tsts = -1;
01040   ctl->qnt_tnat = -1;
01041   ctl->qnt_gw_var = -1;
01042   ctl->qnt_stat = -1;
01043
01044   /* Read quantities... */
01045   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01046   if (ctl->nq > NQ)
01047     ERRMSG("Too many quantities!");
01048   for (iq = 0; iq < ctl->nq; iq++) {
01049
01050     /* Read quantity name and format... */
01051     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01052     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01053              ctl->qnt_format[iq]);
01054
01055     /* Try to identify quantity... */
01056     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01057       ctl->qnt_ens = iq;
01058       sprintf(ctl->qnt_unit[iq], "-");
01059     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01060       ctl->qnt_m = iq;
01061       sprintf(ctl->qnt_unit[iq], "kg");
01062     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01063       ctl->qnt_r = iq;
01064       sprintf(ctl->qnt_unit[iq], "m");
01065     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01066       ctl->qnt_rho = iq;
01067       sprintf(ctl->qnt_unit[iq], "kg/m^3");
01068     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01069       ctl->qnt_ps = iq;
01070       sprintf(ctl->qnt_unit[iq], "hPa");
01071     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01072       ctl->qnt_pt = iq;
01073       sprintf(ctl->qnt_unit[iq], "hPa");
01074     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01075       ctl->qnt_z = iq;
01076       sprintf(ctl->qnt_unit[iq], "km");
01077     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01078       ctl->qnt_p = iq;
01079       sprintf(ctl->qnt_unit[iq], "hPa");
01080     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
```

```
01081       ctl->qnt_t = iq;
01082       sprintf(ctl->qnt_unit[iq], "K");
01083     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01084       ctl->qnt_u = iq;
01085       sprintf(ctl->qnt_unit[iq], "m/s");
01086     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01087       ctl->qnt_v = iq;
01088       sprintf(ctl->qnt_unit[iq], "m/s");
01089     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01090       ctl->qnt_w = iq;
01091       sprintf(ctl->qnt_unit[iq], "hPa/s");
01092     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01093       ctl->qnt_h2o = iq;
01094       sprintf(ctl->qnt_unit[iq], "1");
01095     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01096       ctl->qnt_o3 = iq;
01097       sprintf(ctl->qnt_unit[iq], "1");
01098     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01099       ctl->qnt_theta = iq;
01100       sprintf(ctl->qnt_unit[iq], "K");
01101     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01102       ctl->qnt_pv = iq;
01103       sprintf(ctl->qnt_unit[iq], "PVU");
01104     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01105       ctl->qnt_tice = iq;
01106       sprintf(ctl->qnt_unit[iq], "K");
01107     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01108       ctl->qnt_tsts = iq;
01109       sprintf(ctl->qnt_unit[iq], "K");
01110     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01111       ctl->qnt_tnat = iq;
01112       sprintf(ctl->qnt_unit[iq], "K");
01113     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
01114       ctl->qnt_gw_var = iq;
01115       sprintf(ctl->qnt_unit[iq], "K^2");
01116     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01117       ctl->qnt_stat = iq;
01118       sprintf(ctl->qnt_unit[iq], "-");
01119     } else
01120       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01121   }
01122
01123   /* Time steps of simulation... */
01124   ctl->direction =
01125     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01126   if (ctl->direction != -1 && ctl->direction != 1)
01127     ERRMSG("Set DIRECTION to -1 or 1!");
01128   ctl->t_start =
01129     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
01130   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
01131   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01132
01133   /* Meteorological data... */
01134   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01135   ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01136   ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01137   ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01138   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01139   if (ctl->met_np > EP)
01140     ERRMSG("Too many levels!");
01141   for (ip = 0; ip < ctl->met_np; ip++)
01142     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01143   ctl->met_tropo
01144     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01145   scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01146   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01147
01148   /* Isosurface parameters... */
01149   ctl->isosurf
01150     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01151   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01152
01153   /* Diffusion parameters... */
01154   ctl->turb_dx_trop
01155     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
01156   ctl->turb_dx_strat
01157     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
01158   ctl->turb_dz_trop
01159     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
01160   ctl->turb_dz_strat
01161     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01162   ctl->turb_meso =
01163     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
01164
01165   /* Life time of particles... */
01166   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01167   ctl->tdec_strat =
```

```
01168     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01169
01170   /* PSC analysis... */
01171   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01172   ctl->psc_hno3 =
01173     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01174
01175   /* Gravity wave analysis... */
01176   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
01177
01178   /* Output of atmospheric data... */
01179   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
01180   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01181   ctl->atm_dt_out =
01182     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01183   ctl->atm_filter =
01184     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01185   ctl->atm_type =
01186     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01187
01188   /* Output of CSI data... */
01189   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01190   ctl->csi_dt_out =
01191     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01192   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
01193            ctl->csi_obsfile);
01194   ctl->csi_obsmin =
01195     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01196   ctl->csi_modmin =
01197     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01198   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01199   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01200   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01201   ctl->csi_lon0 =
01202     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01203   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01204   ctl->csi_nx =
01205     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01206   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01207   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01208   ctl->csi_ny =
01209     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01210
01211   /* Output of ensemble data... */
01212   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01213
01214   /* Output of grid data... */
01215   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01216            ctl->grid_basename);
01217   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01218   ctl->grid_dt_out =
01219     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01220   ctl->grid_sparse =
01221     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01222   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01223   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01224   ctl->grid_nz =
01225     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01226   ctl->grid_lon0 =
01227     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01228   ctl->grid_lon1 =
01229     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01230   ctl->grid_nx =
01231     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01232   ctl->grid_lat0 =
01233     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01234   ctl->grid_lat1 =
01235     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01236   ctl->grid_ny =
01237     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01238
01239   /* Output of profile data... */
01240   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01241            ctl->prof_basename);
01242   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
01243   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01244   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01245   ctl->prof_nz =
01246     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01247   ctl->prof_lon0 =
01248     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
```

```
01249   ctl->prof_lon1 =
01250     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01251   ctl->prof_nx =
01252     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01253   ctl->prof_lat0 =
01254     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01255   ctl->prof_lat1 =
01256     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01257   ctl->prof_ny =
01258     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01259
01260   /* Output of station data... */
01261   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01262            ctl->stat_basename);
01263   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01264   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01265   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01266 }
```

Here is the call graph for this function:

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

Read meteorological data file.

Definition at line 1270 of file libtrac.c.

```
01273               {
01274
01275   char cmd[2 * LEN], levname[LEN], tstr[10];
01276
01277   static float help[EX * EY];
01278
01279   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01280
01281   size_t np, nx, ny;
01282
01283   /* Write info... */
01284   printf("Read meteorological data: %s\n", filename);
01285
01286   /* Get time from filename... */
01287   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01288   year = atoi(tstr);
01289   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01290   mon = atoi(tstr);
01291   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01292   day = atoi(tstr);
01293   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01294   hour = atoi(tstr);
01295   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01296
01297   /* Open netCDF file... */
01298   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01299
01300     /* Try to stage meteo file... */
01301     STOP_TIMER(TIMER_INPUT);
01302     START_TIMER(TIMER_STAGE);
01303     if (ctl->met_stage[0] != '-') {
01304       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01305               year, mon, day, hour, filename);
01306       if (system(cmd) != 0)
01307         ERRMSG("Error while staging meteo data!");
```

```
01308     }
01309     STOP_TIMER(TIMER_STAGE);
01310     START_TIMER(TIMER_INPUT);
01311
01312     /* Try to open again... */
01313     NC(nc_open(filename, NC_NOWRITE, &ncid));
01314   }
01315
01316   /* Get dimensions... */
01317   NC(nc_inq_dimid(ncid, "lon", &dimid));
01318   NC(nc_inq_dimlen(ncid, dimid, &nx));
01319   if (nx < 2 || nx > EX)
01320     ERRMSG("Number of longitudes out of range!");
01321
01322   NC(nc_inq_dimid(ncid, "lat", &dimid));
01323   NC(nc_inq_dimlen(ncid, dimid, &ny));
01324   if (ny < 2 || ny > EY)
01325     ERRMSG("Number of latitudes out of range!");
01326
01327   sprintf(levname, "lev");
01328   NC(nc_inq_dimid(ncid, levname, &dimid));
01329   NC(nc_inq_dimlen(ncid, dimid, &np));
01330   if (np == 1) {
01331     sprintf(levname, "lev_2");
01332     NC(nc_inq_dimid(ncid, levname, &dimid));
01333     NC(nc_inq_dimlen(ncid, dimid, &np));
01334   }
01335   if (np < 2 || np > EP)
01336     ERRMSG("Number of levels out of range!");
01337
01338   /* Store dimensions... */
01339   met->np = (int) np;
01340   met->nx = (int) nx;
01341   met->ny = (int) ny;
01342
01343   /* Get horizontal grid... */
01344   NC(nc_inq_varid(ncid, "lon", &varid));
01345   NC(nc_get_var_double(ncid, varid, met->lon));
01346   NC(nc_inq_varid(ncid, "lat", &varid));
01347   NC(nc_get_var_double(ncid, varid, met->lat));
01348
01349   /* Read meteorological data... */
01350   read_met_help(ncid, "t", "T", met, met->t, 1.0);
01351   read_met_help(ncid, "u", "U", met, met->u, 1.0);
01352   read_met_help(ncid, "v", "V", met, met->v, 1.0);
01353   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01354   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
01355   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
01356
01357   /* Meteo data on pressure levels... */
01358   if (ctl->met_np <= 0) {
01359
01360     /* Read pressure levels from file... */
01361     NC(nc_inq_varid(ncid, levname, &varid));
01362     NC(nc_get_var_double(ncid, varid, met->p));
01363     for (ip = 0; ip < met->np; ip++)
01364       met->p[ip] /= 100.;
01365
01366     /* Extrapolate data for lower boundary... */
01367     read_met_extrapolate(met);
01368   }
01369
01370   /* Meteo data on model levels... */
01371   else {
01372
01373     /* Read pressure data from file... */
01374     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01375
01376     /* Interpolate from model levels to pressure levels... */
01377     read_met_ml2pl(ctl, met, met->t);
01378     read_met_ml2pl(ctl, met, met->u);
01379     read_met_ml2pl(ctl, met, met->v);
01380     read_met_ml2pl(ctl, met, met->w);
01381     read_met_ml2pl(ctl, met, met->h2o);
01382     read_met_ml2pl(ctl, met, met->o3);
01383
01384     /* Set pressure levels... */
01385     met->np = ctl->met_np;
01386     for (ip = 0; ip < met->np; ip++)
01387       met->p[ip] = ctl->met_p[ip];
01388   }
01389
01390   /* Check ordering of pressure levels... */
01391   for (ip = 1; ip < met->np; ip++)
01392     if (met->p[ip - 1] < met->p[ip])
01393       ERRMSG("Pressure levels must be descending!");
01394
```

```
01395   /* Read surface pressure... */
01396   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01397       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01398     NC(nc_get_var_float(ncid, varid, help));
01399     for (iy = 0; iy < met->ny; iy++)
01400       for (ix = 0; ix < met->nx; ix++)
01401         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01402   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01403              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01404     NC(nc_get_var_float(ncid, varid, help));
01405     for (iy = 0; iy < met->ny; iy++)
01406       for (ix = 0; ix < met->nx; ix++)
01407         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01408   } else
01409     for (ix = 0; ix < met->nx; ix++)
01410       for (iy = 0; iy < met->ny; iy++)
01411         met->ps[ix][iy] = met->p[0];
01412
01413   /* Create periodic boundary conditions... */
01414   read_met_periodic(met);
01415
01416   /* Calculate geopotential heights... */
01417   read_met_geopot(ctl, met);
01418
01419   /* Calculate tropopause pressure... */
01420   read_met_tropo(ctl, met);
01421
01422   /* Downsampling... */
01423   read_met_sample(ctl, met);
01424
01425   /* Close file... */
01426   NC(nc_close(ncid));
01427 }
```

Here is the call graph for this function:

5.17.2.22 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 1431 of file libtrac.c.

```
01432               {
01433
01434   int ip, ip0, ix, iy;
01435
01436   /* Loop over columns... */
01437   for (ix = 0; ix < met->nx; ix++)
01438     for (iy = 0; iy < met->ny; iy++) {
01439
01440       /* Find lowest valid data point... */
01441       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01442         if (!gsl_finite(met->t[ix][iy][ip0])
01443             || !gsl_finite(met->u[ix][iy][ip0])
01444             || !gsl_finite(met->v[ix][iy][ip0])
01445             || !gsl_finite(met->w[ix][iy][ip0]))
01446           break;
01447
01448       /* Extrapolate... */
01449       for (ip = ip0; ip >= 0; ip--) {
01450         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01451         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01452         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01453         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01454         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01455         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01456       }
01457     }
01458 }
```

5.17.2.23 void read_met_geopot (ctl_t * *ctl*, met_t * *met*)

Calculate geopotential heights.

Definition at line 1462 of file libtrac.c.

```
01464               {
01465
01466   static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01467
01468   static int init, topo_nx = -1, topo_ny;
01469
01470   FILE *in;
01471
01472   char line[LEN];
01473
01474   double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01475
01476   float help[EX][EY];
01477
01478   int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01479
01480   /* Initialize geopotential heights... */
01481   for (ix = 0; ix < met->nx; ix++)
01482     for (iy = 0; iy < met->ny; iy++)
01483       for (ip = 0; ip < met->np; ip++)
01484         met->z[ix][iy][ip] = GSL_NAN;
01485
01486   /* Check filename... */
01487   if (ctl->met_geopot[0] == '-')
01488     return;
01489
01490   /* Read surface geopotential... */
01491   if (!init) {
01492
01493     /* Write info... */
01494     printf("Read surface geopotential: %s\n", ctl->met_geopot);
01495
01496     /* Open file... */
01497     if (!(in = fopen(ctl->met_geopot, "r")))
01498       ERRMSG("Cannot open file!");
01499
01500     /* Read data... */
01501     while (fgets(line, LEN, in))
01502       if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01503         if (rlon != rlon_old) {
01504           if ((++topo_nx) >= EX)
01505             ERRMSG("Too many longitudes!");
01506           topo_ny = 0;
01507         }
01508         rlon_old = rlon;
01509         topo_lon[topo_nx] = rlon;
01510         topo_lat[topo_ny] = rlat;
01511         topo_z[topo_nx][topo_ny] = rz;
01512         if ((++topo_ny) >= EY)
01513           ERRMSG("Too many latitudes!");
01514       }
01515     if ((++topo_nx) >= EX)
01516       ERRMSG("Too many longitudes!");
01517
01518     /* Close file... */
01519     fclose(in);
01520
01521     /* Check grid spacing... */
01522     if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01523         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01524       ERRMSG("Grid spacing does not match!");
01525
01526     /* Set init flag... */
01527     init = 1;
01528   }
01529
01530   /* Apply hydrostatic equation to calculate geopotential heights... */
01531   for (ix = 0; ix < met->nx; ix++)
01532     for (iy = 0; iy < met->ny; iy++) {
01533
01534       /* Get surface height... */
01535       lon = met->lon[ix];
01536       if (lon < topo_lon[0])
01537         lon += 360;
01538       else if (lon > topo_lon[topo_nx - 1])
01539         lon -= 360;
01540       lat = met->lat[iy];
```

```
01541       tx = locate(topo_lon, topo_nx, lon);
01542       ty = locate(topo_lat, topo_ny, lat);
01543       z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01544                topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01545       z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01546                topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01547       z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01548
01549       /* Find surface pressure level... */
01550       ip0 = locate(met->p, met->np, met->ps[ix][iy]);
01551
01552       /* Get surface temperature... */
01553       ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
01554                met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
01555
01556       /* Upper part of profile... */
01557       met->z[ix][iy][ip0 + 1]
01558         = (float) (z0 + 8.31441 / 28.9647 / G0
01559                    * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01560                    * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01561       for (ip = ip0 + 2; ip < met->np; ip++)
01562         met->z[ix][iy][ip]
01563           = (float) (met->z[ix][iy][ip - 1] + 8.31441 / 28.9647 / G0
01564                      * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01565                      * log(met->p[ip - 1] / met->p[ip]));
01566     }
01567
01568   /* Smooth fields... */
01569   for (ip = 0; ip < met->np; ip++) {
01570
01571     /* Median filter... */
01572     for (ix = 0; ix < met->nx; ix++)
01573       for (iy = 0; iy < met->nx; iy++) {
01574         n = 0;
01575         for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
01576           ix3 = ix2;
01577           if (ix3 < 0)
01578             ix3 += met->nx;
01579           if (ix3 >= met->nx)
01580             ix3 -= met->nx;
01581           for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01582                iy2++)
01583             if (gsl_finite(met->z[ix3][iy2][ip])) {
01584               data[n] = met->z[ix3][iy2][ip];
01585               n++;
01586             }
01587         }
01588         if (n > 0) {
01589           gsl_sort(data, 1, (size_t) n);
01590           help[ix][iy] = (float)
01591             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01592         } else
01593           help[ix][iy] = GSL_NAN;
01594       }
01595
01596     /* Copy data... */
01597     for (ix = 0; ix < met->nx; ix++)
01598       for (iy = 0; iy < met->nx; iy++)
01599         met->z[ix][iy][ip] = help[ix][iy];
01600   }
01601 }
```

Here is the call graph for this function:

5.17.2.24 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 1605 of file libtrac.c.

```
01611             {
01612
01613   static float help[EX * EY * EP];
01614
01615   int ip, ix, iy, n = 0, varid;
01616
01617   /* Check if variable exists... */
01618   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01619     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
```

```
01620       return;
01621
01622   /* Read data... */
01623   NC(nc_get_var_float(ncid, varid, help));
01624
01625   /* Copy and check data... */
01626   for (ip = 0; ip < met->np; ip++)
01627     for (iy = 0; iy < met->ny; iy++)
01628       for (ix = 0; ix < met->nx; ix++) {
01629         dest[ix][iy][ip] = scl * help[n++];
01630         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
01631           dest[ix][iy][ip] = GSL_NAN;
01632       }
01633 }
```

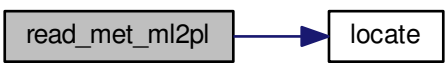
5.17.2.25 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 1637 of file libtrac.c.

```
01640                           {
01641
01642   double aux[EP], p[EP], pt;
01643
01644   int ip, ip2, ix, iy;
01645
01646   /* Loop over columns... */
01647   for (ix = 0; ix < met->nx; ix++)
01648     for (iy = 0; iy < met->ny; iy++) {
01649
01650       /* Copy pressure profile... */
01651       for (ip = 0; ip < met->np; ip++)
01652         p[ip] = met->pl[ix][iy][ip];
01653
01654       /* Interpolate... */
01655       for (ip = 0; ip < ctl->met_np; ip++) {
01656         pt = ctl->met_p[ip];
01657         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01658           pt = p[0];
01659         else if ((pt > p[met->np - 1] && p[1] > p[0])
01660                  || (pt < p[met->np - 1] && p[1] < p[0]))
01661           pt = p[met->np - 1];
01662         ip2 = locate(p, met->np, pt);
01663         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01664                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01665       }
01666
01667       /* Copy data... */
01668       for (ip = 0; ip < ctl->met_np; ip++)
01669         var[ix][iy][ip] = (float) aux[ip];
01670     }
01671 }
```

Here is the call graph for this function:



5.17.2.26 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 1675 of file libtrac.c.

```
01676                {
01677
01678   int ip, iy;
01679
01680   /* Check longitudes... */
01681   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01682              + met->lon[1] - met->lon[0] - 360) < 0.01))
01683     return;
01684
01685   /* Increase longitude counter... */
01686   if ((++met->nx) > EX)
01687     ERRMSG("Cannot create periodic boundary conditions!");
01688
01689   /* Set longitude... */
01690   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01691
01692   /* Loop over latitudes and pressure levels... */
01693   for (iy = 0; iy < met->ny; iy++)
01694     for (ip = 0; ip < met->np; ip++) {
01695       met->ps[met->nx - 1][iy] = met->ps[0][iy];
01696       met->pt[met->nx - 1][iy] = met->pt[0][iy];
01697       met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01698       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01699       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01700       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01701       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01702       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01703       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01704     }
01705 }
```

5.17.2.27 void read_met_sample (ctl_t * *ctl*, met_t * *met*)

Downsampling of meteorological data.

Definition at line 1709 of file libtrac.c.

```
01711                {
01712
01713   met_t *help;
01714
01715   float w, wsum;
01716
01717   int ip, ip2, ix, ix2, iy, iy2;
01718
01719   /* Check parameters... */
01720   if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1)
01721     return;
01722
01723   /* Allocate... */
01724   ALLOC(help, met_t, 1);
01725
01726   /* Copy data... */
01727   help->nx = met->nx;
01728   help->ny = met->ny;
01729   help->np = met->np;
01730   memcpy(help->lon, met->lon, sizeof(met->lon));
01731   memcpy(help->lat, met->lat, sizeof(met->lat));
01732   memcpy(help->p, met->p, sizeof(met->p));
01733
01734   /* Smoothing... */
01735   for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01736     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01737       for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01738         help->ps[ix][iy] = 0;
01739         help->pt[ix][iy] = 0;
01740         help->z[ix][iy][ip] = 0;
01741         help->t[ix][iy][ip] = 0;
01742         help->u[ix][iy][ip] = 0;
```

```
01743         help->v[ix][iy][ip] = 0;
01744         help->w[ix][iy][ip] = 0;
01745         help->h2o[ix][iy][ip] = 0;
01746         help->o3[ix][iy][ip] = 0;
01747         wsum = 0;
01748         for (ix2 = GSL_MAX(ix - ctl->met_dx + 1, 0);
01749              ix2 <= GSL_MIN(ix + ctl->met_dx - 1, met->nx - 1); ix2++)
01750           for (iy2 = GSL_MAX(iy - ctl->met_dy + 1, 0);
01751                iy2 <= GSL_MIN(iy + ctl->met_dy - 1, met->ny - 1); iy2++)
01752             for (ip2 = GSL_MAX(ip - ctl->met_dp + 1, 0);
01753                  ip2 <= GSL_MIN(ip + ctl->met_dp - 1, met->np - 1); ip2++) {
01754               w = (float) (1.0 - fabs(ix - ix2) / ctl->met_dx)
01755                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_dy)
01756                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_dp);
01757               help->ps[ix][iy] += w * met->ps[ix2][iy2];
01758               help->pt[ix][iy] += w * met->pt[ix2][iy2];
01759               help->z[ix][iy][ip] += w * met->z[ix2][iy2][ip2];
01760               help->t[ix][iy][ip] += w * met->t[ix2][iy2][ip2];
01761               help->u[ix][iy][ip] += w * met->u[ix2][iy2][ip2];
01762               help->v[ix][iy][ip] += w * met->v[ix2][iy2][ip2];
01763               help->w[ix][iy][ip] += w * met->w[ix2][iy2][ip2];
01764               help->h2o[ix][iy][ip] += w * met->h2o[ix2][iy2][ip2];
01765               help->o3[ix][iy][ip] += w * met->o3[ix2][iy2][ip2];
01766               wsum += w;
01767             }
01768         help->ps[ix][iy] /= wsum;
01769         help->pt[ix][iy] /= wsum;
01770         help->t[ix][iy][ip] /= wsum;
01771         help->z[ix][iy][ip] /= wsum;
01772         help->u[ix][iy][ip] /= wsum;
01773         help->v[ix][iy][ip] /= wsum;
01774         help->w[ix][iy][ip] /= wsum;
01775         help->h2o[ix][iy][ip] /= wsum;
01776         help->o3[ix][iy][ip] /= wsum;
01777       }
01778     }
01779   }
01780
01781   /* Downsampling... */
01782   met->nx = 0;
01783   for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01784     met->lon[met->nx] = help->lon[ix];
01785     met->ny = 0;
01786     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01787       met->lat[met->ny] = help->lat[iy];
01788       met->ps[met->nx][met->ny] = help->ps[ix][iy];
01789       met->pt[met->nx][met->ny] = help->pt[ix][iy];
01790       met->np = 0;
01791       for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01792         met->p[met->np] = help->p[ip];
01793         met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01794         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01795         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01796         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01797         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01798         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
01799         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
01800         met->np++;
01801       }
01802       met->ny++;
01803     }
01804     met->nx++;
01805   }
01806
01807   /* Free... */
01808   free(help);
01809 }
```

5.17.2.28 void read_met_tropo (ctl_t * *ctl*, met_t * *met*)

Calculate tropopause pressure.

Definition at line 1813 of file libtrac.c.

```
01815               {
01816
01817   gsl_interp_accel *acc;
01818
01819   gsl_spline *spline;
01820
```

```
01821   double tt[400], tt2[400], tz[400], tz2[400];
01822
01823   int found, ix, iy, iz, iz2;
01824
01825   /* Allocate... */
01826   acc = gsl_interp_accel_alloc();
01827   spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
01828
01829   /* Do not calculate tropopause... */
01830   if (ctl->met_tropo == 0)
01831     for (ix = 0; ix < met->nx; ix++)
01832       for (iy = 0; iy < met->ny; iy++)
01833         met->pt[ix][iy] = GSL_NAN;
01834
01835   /* Use tropopause climatology... */
01836   else if (ctl->met_tropo == 1)
01837     for (ix = 0; ix < met->nx; ix++)
01838       for (iy = 0; iy < met->ny; iy++)
01839         met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
01840
01841   /* Use cold point... */
01842   else if (ctl->met_tropo == 2) {
01843
01844     /* Loop over grid points... */
01845     for (ix = 0; ix < met->nx; ix++)
01846       for (iy = 0; iy < met->ny; iy++) {
01847
01848         /* Get temperature profile... */
01849         for (iz = 0; iz < met->np; iz++) {
01850           tz[iz] = Z(met->p[iz]);
01851           tt[iz] = met->t[ix][iy][iz];
01852         }
01853
01854         /* Interpolate temperature profile... */
01855         gsl_spline_init(spline, tz, tt, (size_t) met->np);
01856         for (iz = 0; iz <= 170; iz++) {
01857           tz2[iz] = 4.5 + 0.1 * iz;
01858           tt2[iz] = gsl_spline_eval(spline, tz2[iz], acc);
01859         }
01860
01861         /* Find minimum... */
01862         iz = (int) gsl_stats_min_index(tt2, 1, 171);
01863         if (iz <= 0 || iz >= 170)
01864           met->pt[ix][iy] = GSL_NAN;
01865         else
01866           met->pt[ix][iy] = P(tz2[iz]);
01867       }
01868   }
01869
01870   /* Use WMO definition... */
01871   else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
01872
01873     /* Loop over grid points... */
01874     for (ix = 0; ix < met->nx; ix++)
01875       for (iy = 0; iy < met->ny; iy++) {
01876
01877         /* Get temperature profile... */
01878         for (iz = 0; iz < met->np; iz++) {
01879           tz[iz] = Z(met->p[iz]);
01880           tt[iz] = met->t[ix][iy][iz];
01881         }
01882
01883         /* Interpolate temperature profile... */
01884         gsl_spline_init(spline, tz, tt, (size_t) met->np);
01885         for (iz = 0; iz <= 160; iz++) {
01886           tz2[iz] = 4.5 + 0.1 * iz;
01887           tt2[iz] = gsl_spline_eval(spline, tz2[iz], acc);
01888         }
01889
01890         /* Find 1st tropopause... */
01891         met->pt[ix][iy] = GSL_NAN;
01892         for (iz = 0; iz <= 140; iz++) {
01893           found = 1;
01894           for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
01895             if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01896                 / log(P(tz2[iz2]) / P(tz2[iz])) > 2.0) {
01897               found = 0;
01898               break;
01899             }
01900           if (found) {
01901             if (iz > 0 && iz < 140)
01902               met->pt[ix][iy] = P(tz2[iz]);
01903             break;
01904           }
01905         }
01906
01907         /* Find 2nd tropopause... */
```

```
01908         if (ctl->met_tropo == 4) {
01909           met->pt[ix][iy] = GSL_NAN;
01910           for (; iz <= 140; iz++) {
01911             found = 1;
01912             for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
01913               if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01914                   / log(P(tz2[iz2]) / P(tz2[iz])) < 3.0) {
01915                 found = 0;
01916                 break;
01917               }
01918             if (found)
01919               break;
01920           }
01921           for (; iz <= 140; iz++) {
01922             found = 1;
01923             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
01924               if (1000. * G0 / R0 * log(tt2[iz2] / tt2[iz])
01925                   / log(P(tz2[iz2]) / P(tz2[iz])) > 2.0) {
01926                 found = 0;
01927                 break;
01928               }
01929             if (found) {
01930               if (iz > 0 && iz < 140)
01931                 met->pt[ix][iy] = P(tz2[iz]);
01932               break;
01933             }
01934           }
01935         }
01936       }
01937   }
01938
01939   else
01940     ERRMSG("Cannot calculate tropopause!");
01941
01942   /* Free... */
01943   gsl_spline_free(spline);
01944   gsl_interp_accel_free(acc);
01945 }
```

Here is the call graph for this function:

5.17.2.29 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 1949 of file libtrac.c.

```
01956               {
01957
01958   FILE *in = NULL;
01959
01960   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01961     msg[LEN], rvarname[LEN], rval[LEN];
01962
01963   int contain = 0, i;
01964
01965   /* Open file... */
01966   if (filename[strlen(filename) - 1] != '-')
01967     if (!(in = fopen(filename, "r")))
01968       ERRMSG("Cannot open file!");
01969
01970   /* Set full variable name... */
01971   if (arridx >= 0) {
01972     sprintf(fullname1, "%s[%d]", varname, arridx);
01973     sprintf(fullname2, "%s[*]", varname);
01974   } else {
01975     sprintf(fullname1, "%s", varname);
01976     sprintf(fullname2, "%s", varname);
01977   }
01978
01979   /* Read data... */
01980   if (in != NULL)
01981     while (fgets(line, LEN, in))
01982       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01983         if (strcasecmp(rvarname, fullname1) == 0 ||
01984             strcasecmp(rvarname, fullname2) == 0) {
01985           contain = 1;
01986           break;
```

```
01987         }
01988     for (i = 1; i < argc - 1; i++)
01989       if (strcasecmp(argv[i], fullname1) == 0 ||
01990           strcasecmp(argv[i], fullname2) == 0) {
01991         sprintf(rval, "%s", argv[i + 1]);
01992         contain = 1;
01993         break;
01994       }
01995
01996     /* Close file... */
01997     if (in != NULL)
01998       fclose(in);
01999
02000     /* Check for missing variables... */
02001     if (!contain) {
02002       if (strlen(defvalue) > 0)
02003         sprintf(rval, "%s", defvalue);
02004       else {
02005         sprintf(msg, "Missing variable %s!\n", fullname1);
02006         ERRMSG(msg);
02007       }
02008     }
02009
02010     /* Write info... */
02011     printf("%s = %s\n", fullname1, rval);
02012
02013     /* Return values... */
02014     if (value != NULL)
02015       sprintf(value, "%s", rval);
02016     return atof(rval);
02017 }
```

5.17.2.30 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 2021 of file libtrac.c.

```
02029                 {
02030
02031     struct tm t0, t1;
02032
02033     t0.tm_year = 100;
02034     t0.tm_mon = 0;
02035     t0.tm_mday = 1;
02036     t0.tm_hour = 0;
02037     t0.tm_min = 0;
02038     t0.tm_sec = 0;
02039
02040     t1.tm_year = year - 1900;
02041     t1.tm_mon = mon - 1;
02042     t1.tm_mday = day;
02043     t1.tm_hour = hour;
02044     t1.tm_min = min;
02045     t1.tm_sec = sec;
02046
02047     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02048 }
```

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

Measure wall-clock time.

Definition at line 2052 of file libtrac.c.

```
02055             {
02056
02057     static double starttime[NTIMER], runtime[NTIMER];
02058
02059     /* Check id... */
02060     if (id < 0 || id >= NTIMER)
02061       ERRMSG("Too many timers!");
02062
02063     /* Start timer... */
```

```
02064   if (mode == 1) {
02065     if (starttime[id] <= 0)
02066       starttime[id] = omp_get_wtime();
02067     else
02068       ERRMSG("Timer already started!");
02069   }
02070
02071   /* Stop timer... */
02072   else if (mode == 2) {
02073     if (starttime[id] > 0) {
02074       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02075       starttime[id] = -1;
02076     }
02077   }
02078
02079   /* Print timer... */
02080   else if (mode == 3)
02081     printf("%s = %.3f s\n", name, runtime[id]);
02082 }
```

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

Write atmospheric data.

Definition at line 2086 of file libtrac.c.

```
02090             {
02091
02092   FILE *in, *out;
02093
02094   char line[LEN];
02095
02096   double r, t0, t1;
02097
02098   int ip, iq, year, mon, day, hour, min, sec;
02099
02100   /* Set time interval for output... */
02101   t0 = t - 0.5 * ctl->dt_mod;
02102   t1 = t + 0.5 * ctl->dt_mod;
02103
02104   /* Write info... */
02105   printf("Write atmospheric data: %s\n", filename);
02106
02107   /* Write ASCII data... */
02108   if (ctl->atm_type == 0) {
02109
02110     /* Check if gnuplot output is requested... */
02111     if (ctl->atm_gpfile[0] != '-') {
02112
02113       /* Create gnuplot pipe... */
02114       if (!(out = popen("gnuplot", "w")))
02115         ERRMSG("Cannot create pipe to gnuplot!");
02116
02117       /* Set plot filename... */
02118       fprintf(out, "set out \"%s.png\"\n", filename);
02119
02120       /* Set time string... */
02121       jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02122       fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02123               year, mon, day, hour, min);
02124
02125       /* Dump gnuplot file to pipe... */
02126       if (!(in = fopen(ctl->atm_gpfile, "r")))
02127         ERRMSG("Cannot open file!");
02128       while (fgets(line, LEN, in))
02129         fprintf(out, "%s", line);
02130       fclose(in);
02131     }
02132
02133     else {
02134
02135       /* Create file... */
02136       if (!(out = fopen(filename, "w")))
02137         ERRMSG("Cannot create file!");
02138     }
02139
02140     /* Write header... */
02141     fprintf(out,
02142             "# $1 = time [s]\n"
02143             "# $2 = altitude [km]\n"
```

```
02144             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02145     for (iq = 0; iq < ctl->nq; iq++)
02146       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02147               ctl->qnt_unit[iq]);
02148     fprintf(out, "\n");
02149
02150     /* Write data... */
02151     for (ip = 0; ip < atm->np; ip++) {
02152
02153       /* Check time... */
02154       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02155         continue;
02156
02157       /* Write output... */
02158       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02159               atm->lon[ip], atm->lat[ip]);
02160       for (iq = 0; iq < ctl->nq; iq++) {
02161         fprintf(out, " ");
02162         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02163       }
02164       fprintf(out, "\n");
02165     }
02166
02167     /* Close file... */
02168     fclose(out);
02169   }
02170
02171   /* Write binary data... */
02172   else if (ctl->atm_type == 1) {
02173
02174     /* Create file... */
02175     if (!(out = fopen(filename, "w")))
02176       ERRMSG("Cannot create file!");
02177
02178     /* Write data... */
02179     FWRITE(&atm->np, int,
02180            1,
02181            out);
02182     FWRITE(atm->time, double,
02183              (size_t) atm->np,
02184            out);
02185     FWRITE(atm->p, double,
02186              (size_t) atm->np,
02187            out);
02188     FWRITE(atm->lon, double,
02189              (size_t) atm->np,
02190            out);
02191     FWRITE(atm->lat, double,
02192              (size_t) atm->np,
02193            out);
02194     for (iq = 0; iq < ctl->nq; iq++)
02195       FWRITE(atm->q[iq], double,
02196                (size_t) atm->np,
02197              out);
02198
02199     /* Close file... */
02200     fclose(out);
02201   }
02202
02203   /* Error... */
02204   else
02205     ERRMSG("Atmospheric data type not supported!");
02206 }
```

Here is the call graph for this function:

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

Write CSI data.

Definition at line 2210 of file libtrac.c.

```
02214           {
02215
02216   static FILE *in, *out;
02217
02218   static char line[LEN];
02219
02220   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02221     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02222
02223   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02224
02225   /* Init... */
02226   if (!init) {
02227     init = 1;
02228
02229     /* Check quantity index for mass... */
02230     if (ctl->qnt_m < 0)
02231       ERRMSG("Need quantity mass to analyze CSI!");
02232
02233     /* Open observation data file... */
02234     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02235     if (!(in = fopen(ctl->csi_obsfile, "r")))
02236       ERRMSG("Cannot open file!");
02237
02238     /* Create new file... */
02239     printf("Write CSI data: %s\n", filename);
02240     if (!(out = fopen(filename, "w")))
02241       ERRMSG("Cannot create file!");
02242
02243     /* Write header... */
02244     fprintf(out,
02245             "# $1 = time [s]\n"
02246             "# $2 = number of hits (cx)\n"
02247             "# $3 = number of misses (cy)\n"
02248             "# $4 = number of false alarms (cz)\n"
02249             "# $5 = number of observations (cx + cy)\n"
02250             "# $6 = number of forecasts (cx + cz)\n"
02251             "# $7 = bias (forecasts/observations) [%%]\n"
02252             "# $8 = probability of detection (POD) [%%]\n"
02253             "# $9 = false alarm rate (FAR) [%%]\n"
02254             "# $10 = critical success index (CSI) [%%]\n\n");
02255   }
02256
02257   /* Set time interval... */
02258   t0 = t - 0.5 * ctl->dt_mod;
02259   t1 = t + 0.5 * ctl->dt_mod;
02260
02261   /* Initialize grid cells... */
02262   for (ix = 0; ix < ctl->csi_nx; ix++)
02263     for (iy = 0; iy < ctl->csi_ny; iy++)
02264       for (iz = 0; iz < ctl->csi_nz; iz++)
02265         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02266
02267   /* Read data... */
02268   while (fgets(line, LEN, in)) {
02269
02270     /* Read data... */
02271     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02272         5)
02273       continue;
02274
02275     /* Check time... */
02276     if (rt < t0)
02277       continue;
02278     if (rt > t1)
02279       break;
02280
02281     /* Calculate indices... */
02282     ix = (int) ((rlon - ctl->csi_lon0)
02283                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02284     iy = (int) ((rlat - ctl->csi_lat0)
02285                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02286     iz = (int) ((rz - ctl->csi_z0)
02287                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02288
02289     /* Check indices... */
02290     if (ix < 0 || ix >= ctl->csi_nx ||
02291         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02292       continue;
02293
02294     /* Get mean observation index... */
02295     obsmean[ix][iy][iz] += robs;
02296     obscount[ix][iy][iz]++;
02297   }
```

```
02298
02299   /* Analyze model data... */
02300   for (ip = 0; ip < atm->np; ip++) {
02301
02302     /* Check time... */
02303     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02304       continue;
02305
02306     /* Get indices... */
02307     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02308                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02309     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02310                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02311     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02312                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02313
02314     /* Check indices... */
02315     if (ix < 0 || ix >= ctl->csi_nx ||
02316         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02317       continue;
02318
02319     /* Get total mass in grid cell... */
02320     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02321   }
02322
02323   /* Analyze all grid cells... */
02324   for (ix = 0; ix < ctl->csi_nx; ix++)
02325     for (iy = 0; iy < ctl->csi_ny; iy++)
02326       for (iz = 0; iz < ctl->csi_nz; iz++) {
02327
02328         /* Calculate mean observation index... */
02329         if (obscount[ix][iy][iz] > 0)
02330           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02331
02332         /* Calculate column density... */
02333         if (modmean[ix][iy][iz] > 0) {
02334           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02335           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02336           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02337           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02338             * cos(lat * M_PI / 180.);
02339           modmean[ix][iy][iz] /= (1e6 * area);
02340         }
02341
02342         /* Calculate CSI... */
02343         if (obscount[ix][iy][iz] > 0) {
02344           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02345               modmean[ix][iy][iz] >= ctl->csi_modmin)
02346             cx++;
02347           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02348                    modmean[ix][iy][iz] < ctl->csi_modmin)
02349             cy++;
02350           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02351                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02352             cz++;
02353         }
02354       }
02355
02356   /* Write output... */
02357   if (fmod(t, ctl->csi_dt_out) == 0) {
02358
02359     /* Write... */
02360     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02361             t, cx, cy, cz, cx + cy, cx + cz,
02362             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02363             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02364             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02365             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02366
02367     /* Set counters to zero... */
02368     cx = cy = cz = 0;
02369   }
02370
02371   /* Close file... */
02372   if (t == ctl->t_stop)
02373     fclose(out);
02374 }
```

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

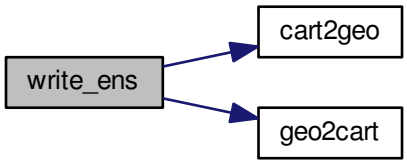
Write ensemble data.

Definition at line 2378 of file libtrac.c.

```
02382           {
02383
02384   static FILE *out;
02385
02386   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02387     t0, t1, x[NENS][3], xm[3];
02388
02389   static int init, ip, iq;
02390
02391   static size_t i, n;
02392
02393   /* Init... */
02394   if (!init) {
02395     init = 1;
02396
02397     /* Check quantities... */
02398     if (ctl->qnt_ens < 0)
02399       ERRMSG("Missing ensemble IDs!");
02400
02401     /* Create new file... */
02402     printf("Write ensemble data: %s\n", filename);
02403     if (!(out = fopen(filename, "w")))
02404       ERRMSG("Cannot create file!");
02405
02406     /* Write header... */
02407     fprintf(out,
02408             "# $1 = time [s]\n"
02409             "# $2 = altitude [km]\n"
02410             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02411     for (iq = 0; iq < ctl->nq; iq++)
02412       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02413               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02414     for (iq = 0; iq < ctl->nq; iq++)
02415       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02416               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02417     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02418   }
02419
02420   /* Set time interval... */
02421   t0 = t - 0.5 * ctl->dt_mod;
02422   t1 = t + 0.5 * ctl->dt_mod;
02423
02424   /* Init... */
02425   ens = GSL_NAN;
02426   n = 0;
02427
02428   /* Loop over air parcels... */
02429   for (ip = 0; ip < atm->np; ip++) {
02430
02431     /* Check time... */
02432     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02433       continue;
02434
02435     /* Check ensemble id... */
02436     if (atm->q[ctl->qnt_ens][ip] != ens) {
02437
02438       /* Write results... */
02439       if (n > 0) {
02440
02441         /* Get mean position... */
02442         xm[0] = xm[1] = xm[2] = 0;
02443         for (i = 0; i < n; i++) {
02444           xm[0] += x[i][0] / (double) n;
02445           xm[1] += x[i][1] / (double) n;
02446           xm[2] += x[i][2] / (double) n;
02447         }
02448         cart2geo(xm, &dummy, &lon, &lat);
02449         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02450                 lat);
02451
02452         /* Get quantity statistics... */
02453         for (iq = 0; iq < ctl->nq; iq++) {
02454           fprintf(out, " ");
02455           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02456         }
02457         for (iq = 0; iq < ctl->nq; iq++) {
02458           fprintf(out, " ");
02459           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02460         }
02461         fprintf(out, " %lu\n", n);
02462       }
02463
02464       /* Init new ensemble... */
02465       ens = atm->q[ctl->qnt_ens][ip];
02466       n = 0;
02467     }
02468
```

```
02469     /* Save data... */
02470     p[n] = atm->p[ip];
02471     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02472     for (iq = 0; iq < ctl->nq; iq++)
02473       q[iq][n] = atm->q[iq][ip];
02474     if ((++n) >= NENS)
02475       ERRMSG("Too many data points!");
02476   }
02477
02478   /* Write results... */
02479   if (n > 0) {
02480
02481     /* Get mean position... */
02482     xm[0] = xm[1] = xm[2] = 0;
02483     for (i = 0; i < n; i++) {
02484       xm[0] += x[i][0] / (double) n;
02485       xm[1] += x[i][1] / (double) n;
02486       xm[2] += x[i][2] / (double) n;
02487     }
02488     cart2geo(xm, &dummy, &lon, &lat);
02489     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02490
02491     /* Get quantity statistics... */
02492     for (iq = 0; iq < ctl->nq; iq++) {
02493       fprintf(out, " ");
02494       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02495     }
02496     for (iq = 0; iq < ctl->nq; iq++) {
02497       fprintf(out, " ");
02498       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02499     }
02500     fprintf(out, " %lu\n", n);
02501   }
02502
02503   /* Close file... */
02504   if (t == ctl->t_stop)
02505     fclose(out);
02506 }
```

Here is the call graph for this function:



5.17.2.35 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 2510 of file libtrac.c.

```
02516             {
02517
02518   FILE *in, *out;
02519
02520   char line[LEN];
02521
02522   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02523     area, rho_air, press, temp, cd, mmr, t0, t1, r;
02524
02525   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
02526
```

```
02527   /* Check dimensions... */
02528   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02529     ERRMSG("Grid dimensions too large!");
02530
02531   /* Check quantity index for mass... */
02532   if (ctl->qnt_m < 0)
02533     ERRMSG("Need quantity mass to write grid data!");
02534
02535   /* Set time interval for output... */
02536   t0 = t - 0.5 * ctl->dt_mod;
02537   t1 = t + 0.5 * ctl->dt_mod;
02538
02539   /* Set grid box size... */
02540   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02541   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02542   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02543
02544   /* Initialize grid... */
02545   for (ix = 0; ix < ctl->grid_nx; ix++)
02546     for (iy = 0; iy < ctl->grid_ny; iy++)
02547       for (iz = 0; iz < ctl->grid_nz; iz++)
02548         grid_m[ix][iy][iz] = 0;
02549
02550   /* Average data... */
02551   for (ip = 0; ip < atm->np; ip++)
02552     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02553
02554       /* Get index... */
02555       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02556       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02557       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02558
02559       /* Check indices... */
02560       if (ix < 0 || ix >= ctl->grid_nx ||
02561           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02562         continue;
02563
02564       /* Add mass... */
02565       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02566     }
02567
02568   /* Check if gnuplot output is requested... */
02569   if (ctl->grid_gpfile[0] != '-') {
02570
02571     /* Write info... */
02572     printf("Plot grid data: %s.png\n", filename);
02573
02574     /* Create gnuplot pipe... */
02575     if (!(out = popen("gnuplot", "w")))
02576       ERRMSG("Cannot create pipe to gnuplot!");
02577
02578     /* Set plot filename... */
02579     fprintf(out, "set out \"%s.png\"\n", filename);
02580
02581     /* Set time string... */
02582     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02583     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02584             year, mon, day, hour, min);
02585
02586     /* Dump gnuplot file to pipe... */
02587     if (!(in = fopen(ctl->grid_gpfile, "r")))
02588       ERRMSG("Cannot open file!");
02589     while (fgets(line, LEN, in))
02590       fprintf(out, "%s", line);
02591     fclose(in);
02592   }
02593
02594   else {
02595
02596     /* Write info... */
02597     printf("Write grid data: %s\n", filename);
02598
02599     /* Create file... */
02600     if (!(out = fopen(filename, "w")))
02601       ERRMSG("Cannot create file!");
02602   }
02603
02604   /* Write header... */
02605   fprintf(out,
02606           "# $1 = time [s]\n"
02607           "# $2 = altitude [km]\n"
02608           "# $3 = longitude [deg]\n"
02609           "# $4 = latitude [deg]\n"
02610           "# $5 = surface area [km^2]\n"
02611           "# $6 = layer width [km]\n"
02612           "# $7 = temperature [K]\n"
02613           "# $8 = column density [kg/m^2]\n"
```

```
02614           "# $9 = mass mixing ratio [1]\n\n");
02615
02616   /* Write data... */
02617   for (ix = 0; ix < ctl->grid_nx; ix++) {
02618     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02619       fprintf(out, "\n");
02620     for (iy = 0; iy < ctl->grid_ny; iy++) {
02621       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02622         fprintf(out, "\n");
02623       for (iz = 0; iz < ctl->grid_nz; iz++)
02624         if (!ctl->grid_sparse
02625             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
02626
02627           /* Set coordinates... */
02628           z = ctl->grid_z0 + dz * (iz + 0.5);
02629           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02630           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02631
02632           /* Get pressure and temperature... */
02633           press = P(z);
02634           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02635                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
02636
02637           /* Calculate surface area... */
02638           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
02639             * cos(lat * M_PI / 180.);
02640
02641           /* Calculate column density... */
02642           cd = grid_m[ix][iy][iz] / (1e6 * area);
02643
02644           /* Calculate mass mixing ratio... */
02645           rho_air = 100. * press / (R0 * temp);
02646           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
02647
02648           /* Write output... */
02649           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
02650                   t, z, lon, lat, area, dz, temp, cd, mmr);
02651         }
02652     }
02653   }
02654
02655   /* Close file... */
02656   fclose(out);
02657 }
```

Here is the call graph for this function:

5.17.2.36 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 2661 of file libtrac.c.

```
02667           {
02668
02669   static FILE *in, *out;
02670
02671   static char line[LEN];
02672
02673   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
02674     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
02675     press, temp, rho_air, mmr, h2o, o3;
02676
02677   static int init, obscount[GX][GY], ip, ix, iy, iz;
02678
02679   /* Init... */
02680   if (!init) {
02681     init = 1;
02682
02683     /* Check quantity index for mass... */
02684     if (ctl->qnt_m < 0)
02685       ERRMSG("Need quantity mass!");
02686
02687     /* Check dimensions... */
02688     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02689       ERRMSG("Grid dimensions too large!");
02690
02691     /* Open observation data file... */
02692     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
```

```
02693     if (!(in = fopen(ctl->prof_obsfile, "r")))
02694       ERRMSG("Cannot open file!");
02695
02696     /* Create new file... */
02697     printf("Write profile data: %s\n", filename);
02698     if (!(out = fopen(filename, "w")))
02699       ERRMSG("Cannot create file!");
02700
02701     /* Write header... */
02702     fprintf(out,
02703             "# $1  = time [s]\n"
02704             "# $2  = altitude [km]\n"
02705             "# $3  = longitude [deg]\n"
02706             "# $4  = latitude [deg]\n"
02707             "# $5  = pressure [hPa]\n"
02708             "# $6  = temperature [K]\n"
02709             "# $7  = mass mixing ratio [1]\n"
02710             "# $8  = H2O volume mixing ratio [1]\n"
02711             "# $9  = O3 volume mixing ratio [1]\n"
02712             "# $10 = mean BT index [K]\n");
02713
02714     /* Set grid box size... */
02715     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
02716     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02717     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02718   }
02719
02720   /* Set time interval... */
02721   t0 = t - 0.5 * ctl->dt_mod;
02722   t1 = t + 0.5 * ctl->dt_mod;
02723
02724   /* Initialize... */
02725   for (ix = 0; ix < ctl->prof_nx; ix++)
02726     for (iy = 0; iy < ctl->prof_ny; iy++) {
02727       obsmean[ix][iy] = 0;
02728       obscount[ix][iy] = 0;
02729       tmean[ix][iy] = 0;
02730       for (iz = 0; iz < ctl->prof_nz; iz++)
02731         mass[ix][iy][iz] = 0;
02732     }
02733
02734   /* Read data... */
02735   while (fgets(line, LEN, in)) {
02736
02737     /* Read data... */
02738     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
02739       continue;
02740
02741     /* Check time... */
02742     if (rt < t0)
02743       continue;
02744     if (rt > t1)
02745       break;
02746
02747     /* Calculate indices... */
02748     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
02749     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02750
02751     /* Check indices... */
02752     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02753       continue;
02754
02755     /* Get mean observation index... */
02756     obsmean[ix][iy] += robs;
02757     tmean[ix][iy] += rt;
02758     obscount[ix][iy]++;
02759   }
02760
02761   /* Analyze model data... */
02762   for (ip = 0; ip < atm->np; ip++) {
02763
02764     /* Check time... */
02765     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02766       continue;
02767
02768     /* Get indices... */
02769     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
02770     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
02771     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
02772
02773     /* Check indices... */
02774     if (ix < 0 || ix >= ctl->prof_nx ||
02775         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
02776       continue;
02777
02778     /* Get total mass in grid cell... */
02779     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
```

```
02780   }
02781
02782   /* Extract profiles... */
02783   for (ix = 0; ix < ctl->prof_nx; ix++)
02784     for (iy = 0; iy < ctl->prof_ny; iy++)
02785       if (obscount[ix][iy] > 0) {
02786
02787         /* Write output... */
02788         fprintf(out, "\n");
02789
02790         /* Loop over altitudes... */
02791         for (iz = 0; iz < ctl->prof_nz; iz++) {
02792
02793           /* Set coordinates... */
02794           z = ctl->prof_z0 + dz * (iz + 0.5);
02795           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
02796           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
02797
02798           /* Get meteorological data... */
02799           press = P(z);
02800           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02801                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02802
02803           /* Calculate mass mixing ratio... */
02804           rho_air = 100. * press / (R0 * temp);
02805           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
02806             * cos(lat * M_PI / 180.);
02807           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02808
02809           /* Write output... */
02810           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02811                   tmean[ix][iy] / obscount[ix][iy],
02812                   z, lon, lat, press, temp, mmr, h2o, o3,
02813                   obsmean[ix][iy] / obscount[ix][iy]);
02814         }
02815       }
02816
02817   /* Close file... */
02818   if (t == ctl->t_stop)
02819     fclose(out);
02820 }
```

Here is the call graph for this function:

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

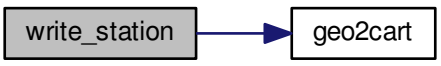
Write station data.

Definition at line 2824 of file libtrac.c.

```
02828             {
02829
02830   static FILE *out;
02831
02832   static double rmax2, t0, t1, x0[3], x1[3];
02833
02834   static int init, ip, iq;
02835
02836   /* Init... */
02837   if (!init) {
02838     init = 1;
02839
02840     /* Write info... */
02841     printf("Write station data: %s\n", filename);
02842
02843     /* Create new file... */
02844     if (!(out = fopen(filename, "w")))
02845       ERRMSG("Cannot create file!");
02846
02847     /* Write header... */
02848     fprintf(out,
02849             "# $1 = time [s]\n"
02850             "# $2 = altitude [km]\n"
02851             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02852     for (iq = 0; iq < ctl->nq; iq++)
02853       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
02854               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02855     fprintf(out, "\n");
02856
```

```
02857     /* Set geolocation and search radius... */
02858     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02859     rmax2 = gsl_pow_2(ctl->stat_r);
02860   }
02861
02862   /* Set time interval for output... */
02863   t0 = t - 0.5 * ctl->dt_mod;
02864   t1 = t + 0.5 * ctl->dt_mod;
02865
02866   /* Loop over air parcels... */
02867   for (ip = 0; ip < atm->np; ip++) {
02868
02869     /* Check time... */
02870     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02871       continue;
02872
02873     /* Check station flag... */
02874     if (ctl->qnt_stat >= 0)
02875       if (atm->q[ctl->qnt_stat][ip])
02876         continue;
02877
02878     /* Get Cartesian coordinates... */
02879     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02880
02881     /* Check horizontal distance... */
02882     if (DIST2(x0, x1) > rmax2)
02883       continue;
02884
02885     /* Set station flag... */
02886     if (ctl->qnt_stat >= 0)
02887       atm->q[ctl->qnt_stat][ip] = 1;
02888
02889     /* Write data... */
02890     fprintf(out, "%.2f %g %g %g",
02891             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02892     for (iq = 0; iq < ctl->nq; iq++) {
02893       fprintf(out, " ");
02894       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02895     }
02896     fprintf(out, "\n");
02897   }
02898
02899   /* Close file... */
02900   if (t == ctl->t_stop)
02901     fclose(out);
02902 }
```

Here is the call graph for this function:



5.18 libtrac.h

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
```

```
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2018 Forschungszentrum Juelich GmbH
00018 */
00019
00034 #include <ctype.h>
00035 #include <gsl/gsl_const_mksa.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_spline.h>
00041 #include <gsl/gsl_statistics.h>
00042 #include <math.h>
00043 #include <netcdf.h>
00044 #include <omp.h>
00045 #include <stdio.h>
00046 #include <stdlib.h>
00047 #include <string.h>
00048 #include <time.h>
00049 #include <sys/time.h>
00050
00051 /* ------------------------------------------------------------
00052    Constants...
00053    ------------------------------------------------------------ */
00054
00056 #define G0 9.80665
00057
00059 #define H0 7.0
00060
00062 #define KB 1.3806504e-23
00063
00065 #define R0 287.058
00066
00068 #define RE 6367.421
00069
00070 /* ------------------------------------------------------------
00071    Dimensions...
00072    ------------------------------------------------------------ */
00073
00075 #define LEN 5000
00076
00078 #define NP 10000000
00079
00081 #define NQ 12
00082
00084 #define EP 111
00085
00087 #define EX 1201
00088
00090 #define EY 601
00091
00093 #define GX 720
00094
00096 #define GY 360
00097
00099 #define GZ 100
00100
00102 #define NENS 2000
00103
00105 #define NTHREADS 512
00106
00107 /* ------------------------------------------------------------
00108    Macros...
00109    ------------------------------------------------------------ */
00110
00112 #define ALLOC(ptr, type, n)                             \
00113   if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)     \
00114     ERRMSG("Out of memory!");
00115
00117 #define DIST(a, b) sqrt(DIST2(a, b))
00118
00120 #define DIST2(a, b)                                                     \
00121   ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00122
00124 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00125
00127 #define ERRMSG(msg) {                                           \
00128     printf("\nError (%s, %s, l%d): %s\n\n",                     \
00129            __FILE__, __func__, __LINE__, msg);                  \
00130     exit(EXIT_FAILURE);                                         \
00131   }
00132
00134 #define FREAD(ptr, type, size, out) {                           \
00135     if(fread(ptr, sizeof(type), size, out)!=size)               \
00136       ERRMSG("Error while reading!");                           \
00137   }
```

```
00138
00140 #define FWRITE(ptr, type, size, out) {                                  \
00141     if(fwrite(ptr, sizeof(type), size, out)!=size)                      \
00142       ERRMSG("Error while writing!");                                   \
00143   }
00144
00146 #define LIN(x0, y0, x1, y1, x)                  \
00147   ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00148
00150 #define NC(cmd) {                                       \
00151     if((cmd)!=NC_NOERR)                                 \
00152       ERRMSG(nc_strerror(cmd));                         \
00153   }
00154
00156 #define NORM(a) sqrt(DOTP(a, a))
00157
00159 #define PRINT(format, var)                                              \
00160   printf("Print (%s, %s, l%d): %s= "format"\n",                         \
00161          __FILE__, __func__, __LINE__, #var, var);
00162
00164 #define P(z) (1013.25*exp(-(z)/H0))
00165
00167 #define THETA(p, t) ((t)*pow(1000./(p), 0.286))
00168
00170 #define TOK(line, tok, format, var) {                                   \
00171     if(((tok)=strtok((line), " \t"))) {                                 \
00172       if(sscanf(tok, format, &(var))!=1) continue;                      \
00173     } else ERRMSG("Error while reading!");                              \
00174   }
00175
00177 #define Z(p) (H0*log(1013.25/(p)))
00178
00179 /* ------------------------------------------------------------
00180    Timers...
00181    ------------------------------------------------------------ */
00182
00184 #define START_TIMER(id) timer(#id, id, 1)
00185
00187 #define STOP_TIMER(id) timer(#id, id, 2)
00188
00190 #define PRINT_TIMER(id) timer(#id, id, 3)
00191
00193 #define NTIMER 13
00194
00196 #define TIMER_TOTAL 0
00197
00199 #define TIMER_INIT 1
00200
00202 #define TIMER_STAGE 2
00203
00205 #define TIMER_INPUT 3
00206
00208 #define TIMER_OUTPUT 4
00209
00211 #define TIMER_ADVECT 5
00212
00214 #define TIMER_DECAY 6
00215
00217 #define TIMER_DIFFMESO 7
00218
00220 #define TIMER_DIFFTURB 8
00221
00223 #define TIMER_ISOSURF 9
00224
00226 #define TIMER_METEO 10
00227
00229 #define TIMER_POSITION 11
00230
00232 #define TIMER_SEDI 12
00233
00234 /* ------------------------------------------------------------
00235    Structs...
00236    ------------------------------------------------------------ */
00237
00239 typedef struct {
00240
00242   int nq;
00243
00245   char qnt_name[NQ][LEN];
00246
00248   char qnt_unit[NQ][LEN];
00249
00251   char qnt_format[NQ][LEN];
00252
00254   int qnt_ens;
00255
00257   int qnt_m;
```

```
00258
00260   int qnt_rho;
00261
00263   int qnt_r;
00264
00266   int qnt_ps;
00267
00269   int qnt_pt;
00270
00272   int qnt_z;
00273
00275   int qnt_p;
00276
00278   int qnt_t;
00279
00281   int qnt_u;
00282
00284   int qnt_v;
00285
00287   int qnt_w;
00288
00290   int qnt_h2o;
00291
00293   int qnt_o3;
00294
00296   int qnt_theta;
00297
00299   int qnt_pv;
00300
00302   int qnt_tice;
00303
00305   int qnt_tsts;
00306
00308   int qnt_tnat;
00309
00311   int qnt_stat;
00312
00314   int qnt_gw_var;
00315
00317   int direction;
00318
00320   double t_start;
00321
00323   double t_stop;
00324
00326   double dt_mod;
00327
00329   double dt_met;
00330
00332   int met_dx;
00333
00335   int met_dy;
00336
00338   int met_dp;
00339
00341   int met_np;
00342
00344   double met_p[EP];
00345
00348   int met_tropo;
00349
00351   char met_geopot[LEN];
00352
00354   char met_stage[LEN];
00355
00358   int isosurf;
00359
00361   char balloon[LEN];
00362
00364   double turb_dx_trop;
00365
00367   double turb_dx_strat;
00368
00370   double turb_dz_trop;
00371
00373   double turb_dz_strat;
00374
00376   double turb_meso;
00377
00379   double tdec_trop;
00380
00382   double tdec_strat;
00383
00385   double psc_h2o;
00386
00388   double psc_hno3;
00389
```

```
00391   char gw_basename[LEN];
00392
00394   char atm_basename[LEN];
00395
00397   char atm_gpfile[LEN];
00398
00400   double atm_dt_out;
00401
00403   int atm_filter;
00404
00406   int atm_type;
00407
00409   char csi_basename[LEN];
00410
00412   double csi_dt_out;
00413
00415   char csi_obsfile[LEN];
00416
00418   double csi_obsmin;
00419
00421   double csi_modmin;
00422
00424   int csi_nz;
00425
00427   double csi_z0;
00428
00430   double csi_z1;
00431
00433   int csi_nx;
00434
00436   double csi_lon0;
00437
00439   double csi_lon1;
00440
00442   int csi_ny;
00443
00445   double csi_lat0;
00446
00448   double csi_lat1;
00449
00451   char grid_basename[LEN];
00452
00454   char grid_gpfile[LEN];
00455
00457   double grid_dt_out;
00458
00460   int grid_sparse;
00461
00463   int grid_nz;
00464
00466   double grid_z0;
00467
00469   double grid_z1;
00470
00472   int grid_nx;
00473
00475   double grid_lon0;
00476
00478   double grid_lon1;
00479
00481   int grid_ny;
00482
00484   double grid_lat0;
00485
00487   double grid_lat1;
00488
00490   char prof_basename[LEN];
00491
00493   char prof_obsfile[LEN];
00494
00496   int prof_nz;
00497
00499   double prof_z0;
00500
00502   double prof_z1;
00503
00505   int prof_nx;
00506
00508   double prof_lon0;
00509
00511   double prof_lon1;
00512
00514   int prof_ny;
00515
00517   double prof_lat0;
00518
00520   double prof_lat1;
```

```
00521
00523   char ens_basename[LEN];
00524
00526   char stat_basename[LEN];
00527
00529   double stat_lon;
00530
00532   double stat_lat;
00533
00535   double stat_r;
00536
00537 } ctl_t;
00538
00540 typedef struct {
00541
00543   int np;
00544
00546   double time[NP];
00547
00549   double p[NP];
00550
00552   double lon[NP];
00553
00555   double lat[NP];
00556
00558   double q[NQ][NP];
00559
00561   float up[NP];
00562
00564   float vp[NP];
00565
00567   float wp[NP];
00568
00569 } atm_t;
00570
00572 typedef struct {
00573
00575   double time;
00576
00578   int nx;
00579
00581   int ny;
00582
00584   int np;
00585
00587   double lon[EX];
00588
00590   double lat[EY];
00591
00593   double p[EP];
00594
00596   double ps[EX][EY];
00597
00599   double pt[EX][EY];
00600
00602   float pl[EX][EY][EP];
00603
00605   float z[EX][EY][EP];
00606
00608   float t[EX][EY][EP];
00609
00611   float u[EX][EY][EP];
00612
00614   float v[EX][EY][EP];
00615
00617   float w[EX][EY][EP];
00618
00620   float h2o[EX][EY][EP];
00621
00623   float o3[EX][EY][EP];
00624
00625 } met_t;
00626
00627 /* ------------------------------------------------------------
00628    Functions...
00629    ------------------------------------------------------------ */
00630
00632 void cart2geo(
00633   double *x,
00634   double *z,
00635   double *lon,
00636   double *lat);
00637
00639 double clim_hno3(
00640   double t,
00641   double lat,
00642   double p);
```

```
00643
00645 double clim_tropo(
00646   double t,
00647   double lat);
00648
00650 double deg2dx(
00651   double dlon,
00652   double lat);
00653
00655 double deg2dy(
00656   double dlat);
00657
00659 double dp2dz(
00660   double dp,
00661   double p);
00662
00664 double dx2deg(
00665   double dx,
00666   double lat);
00667
00669 double dy2deg(
00670   double dy);
00671
00673 double dz2dp(
00674   double dz,
00675   double p);
00676
00678 void geo2cart(
00679   double z,
00680   double lon,
00681   double lat,
00682   double *x);
00683
00685 void get_met(
00686   ctl_t * ctl,
00687   char *metbase,
00688   double t,
00689   met_t * met0,
00690   met_t * met1);
00691
00693 void get_met_help(
00694   double t,
00695   int direct,
00696   char *metbase,
00697   double dt_met,
00698   char *filename);
00699
00701 void intpol_met_2d(
00702   double array[EX][EY],
00703   int ix,
00704   int iy,
00705   double wx,
00706   double wy,
00707   double *var);
00708
00710 void intpol_met_3d(
00711   float array[EX][EY][EP],
00712   int ip,
00713   int ix,
00714   int iy,
00715   double wp,
00716   double wx,
00717   double wy,
00718   double *var);
00719
00721 void intpol_met_space(
00722   met_t * met,
00723   double p,
00724   double lon,
00725   double lat,
00726   double *ps,
00727   double *pt,
00728   double *z,
00729   double *t,
00730   double *u,
00731   double *v,
00732   double *w,
00733   double *h2o,
00734   double *o3);
00735
00737 void intpol_met_time(
00738   met_t * met0,
00739   met_t * met1,
00740   double ts,
00741   double p,
00742   double lon,
00743   double lat,
```

```
00744   double *ps,
00745   double *pt,
00746   double *z,
00747   double *t,
00748   double *u,
00749   double *v,
00750   double *w,
00751   double *h2o,
00752   double *o3);
00753
00755 void jsec2time(
00756   double jsec,
00757   int *year,
00758   int *mon,
00759   int *day,
00760   int *hour,
00761   int *min,
00762   int *sec,
00763   double *remain);
00764
00766 int locate(
00767   double *xx,
00768   int n,
00769   double x);
00770
00772 void read_atm(
00773   const char *filename,
00774   ctl_t * ctl,
00775   atm_t * atm);
00776
00778 void read_ctl(
00779   const char *filename,
00780   int argc,
00781   char *argv[],
00782   ctl_t * ctl);
00783
00785 void read_met(
00786   ctl_t * ctl,
00787   char *filename,
00788   met_t * met);
00789
00791 void read_met_extrapolate(
00792   met_t * met);
00793
00795 void read_met_geopot(
00796   ctl_t * ctl,
00797   met_t * met);
00798
00800 void read_met_help(
00801   int ncid,
00802   char *varname,
00803   char *varname2,
00804   met_t * met,
00805   float dest[EX][EY][EP],
00806   float scl);
00807
00809 void read_met_ml2pl(
00810   ctl_t * ctl,
00811   met_t * met,
00812   float var[EX][EY][EP]);
00813
00815 void read_met_periodic(
00816   met_t * met);
00817
00819 void read_met_sample(
00820   ctl_t * ctl,
00821   met_t * met);
00822
00824 void read_met_tropo(
00825   ctl_t * ctl,
00826   met_t * met);
00827
00829 double scan_ctl(
00830   const char *filename,
00831   int argc,
00832   char *argv[],
00833   const char *varname,
00834   int arridx,
00835   const char *defvalue,
00836   char *value);
00837
00839 void time2jsec(
00840   int year,
00841   int mon,
00842   int day,
00843   int hour,
00844   int min,
```

```
00845   int sec,
00846   double remain,
00847   double *jsec);
00848
00850 void timer(
00851   const char *name,
00852   int id,
00853   int mode);
00854
00856 void write_atm(
00857   const char *filename,
00858   ctl_t * ctl,
00859   atm_t * atm,
00860   double t);
00861
00863 void write_csi(
00864   const char *filename,
00865   ctl_t * ctl,
00866   atm_t * atm,
00867   double t);
00868
00870 void write_ens(
00871   const char *filename,
00872   ctl_t * ctl,
00873   atm_t * atm,
00874   double t);
00875
00877 void write_grid(
00878   const char *filename,
00879   ctl_t * ctl,
00880   met_t * met0,
00881   met_t * met1,
00882   atm_t * atm,
00883   double t);
00884
00886 void write_prof(
00887   const char *filename,
00888   ctl_t * ctl,
00889   met_t * met0,
00890   met_t * met1,
00891   atm_t * atm,
00892   double t);
00893
00895 void write_station(
00896   const char *filename,
00897   ctl_t * ctl,
00898   atm_t * atm,
00899   double t);
```

5.19 match.c File Reference

Calculate deviations between two trajectories.

Functions

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

5.19.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

5.19.2 Function Documentation

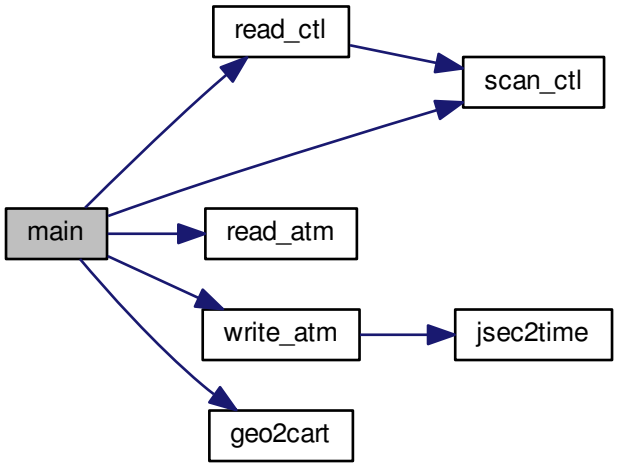
5.19.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file match.c.

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

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

Here is the call graph for this function:



5.20 match.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2, *atm3;
00034
00035   FILE *out;
00036
00037   char filename[LEN];
00038
00039   double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00040
00041   int filter, ip1, ip2, iq, n;
00042
00043   /* Allocate... */
00044   ALLOC(atm1, atm_t, 1);
00045   ALLOC(atm2, atm_t, 1);
00046   ALLOC(atm3, atm_t, 1);
00047
00048   /* Check arguments... */
00049   if (argc < 5)
```

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

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

5.21 met_map.c File Reference

Extract global map from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.21.1 Detailed Description

Extract global map from meteorological data.

Definition in file met_map.c.

5.21.2 Function Documentation

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

Definition at line 27 of file met_map.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *out;
00036
00037   static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], ptm[EX][EY],
00038     tm[EX][EY], um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY],
00039     zm[EX][EY], zt, ztm[EX][EY], tt, ttm[EX][EY];
00040
00041   static int i, ip, ip2, ix, iy, np[EX][EY];
00042
00043   /* Allocate... */
00044   ALLOC(met, met_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00053
00054   /* Loop over files... */
00055   for (i = 3; i < argc; i++) {
00056
00057     /* Read meteorological data... */
00058     read_met(&ctl, argv[i], met);
00059
00060     /* Find nearest pressure level... */
00061     for (ip2 = 0; ip2 < met->np; ip2++) {
00062       dz = fabs(Z(met->p[ip2]) - z);
00063       if (dz < dzmin) {
00064         dzmin = dz;
00065         ip = ip2;
00066       }
00067     }
00068
00069     /* Average data... */
00070     for (ix = 0; ix < met->nx; ix++)
00071       for (iy = 0; iy < met->ny; iy++) {
00072         intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00073                          NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL);
00074         timem[ix][iy] += met->time;
00075         zm[ix][iy] += met->z[ix][iy][ip];
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         ptm[ix][iy] += met->pt[ix][iy];
00084         ztm[ix][iy] += zt;
00085         ttm[ix][iy] += tt;
00086         np[ix][iy]++;
00087       }
00088   }
00089
00090   /* Create output file... */
00091   printf("Write meteorological data file: %s\n", argv[2]);
00092   if (!(out = fopen(argv[2], "w")))
00093     ERRMSG("Cannot create file!");
00094
00095   /* Write header... */
00096   fprintf(out,
00097           "# $1  = time [s]\n"
00098           "# $2  = altitude [km]\n"
00099           "# $3  = longitude [deg]\n"
00100           "# $4  = latitude [deg]\n"
00101           "# $5  = pressure [hPa]\n"
00102           "# $6  = temperature [K]\n"
00103           "# $7  = zonal wind [m/s]\n"
00104           "# $8  = meridional wind [m/s]\n"
00105           "# $9  = vertical wind [hPa/s]\n"
00106           "# $10 = H2O volume mixing ratio [1]\n");
00107   fprintf(out,
00108           "# $11 = O3 volume mixing ratio [1]\n"
00109           "# $12 = geopotential height [km]\n"
00110           "# $13 = surface pressure [hPa]\n"
00111           "# $14 = tropopause pressure [hPa]\n"
00112           "# $15 = tropopause geopotential height [km]\n"
00113           "# $16 = tropopause temperature [K]\n");
00114
```

```
00115   /* Write data... */
00116   for (iy = 0; iy < met->ny; iy++) {
00117     fprintf(out, "\n");
00118     for (ix = 0; ix < met->nx; ix++)
00119       if (met->lon[ix] >= 180)
00120         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00121                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00122                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00123                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00124                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00125                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00126                 zm[ix][iy] / np[ix][iy], psm[ix][iy] / np[ix][iy],
00127                 ptm[ix][iy] / np[ix][iy], ztm[ix][iy] / np[ix][iy],
00128                 ttm[ix][iy] / np[ix][iy]);
00129     for (ix = 0; ix < met->nx; ix++)
00130       if (met->lon[ix] <= 180)
00131         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00132                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00133                 met->lon[ix], met->lat[iy], met->p[ip],
00134                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00135                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00136                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00137                 zm[ix][iy] / np[ix][iy], psm[ix][iy] / np[ix][iy],
00138                 ptm[ix][iy] / np[ix][iy], ztm[ix][iy] / np[ix][iy],
00139                 ttm[ix][iy] / np[ix][iy]);
00140   }
00141
00142   /* Close file... */
00143   fclose(out);
00144
00145   /* Free... */
00146   free(met);
00147
00148   return EXIT_SUCCESS;
00149 }
```

Here is the call graph for this function:

5.22 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   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *out;
00036
00037   static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], ptm[EX][EY],
00038     tm[EX][EY], um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY],
00039     zm[EX][EY], zt, ztm[EX][EY], tt, ttm[EX][EY];
00040
00041   static int i, ip, ip2, ix, iy, np[EX][EY];
00042
00043   /* Allocate... */
00044   ALLOC(met, met_t, 1);
00045
00046   /* Check arguments... */
```

```
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00053
00054   /* Loop over files... */
00055   for (i = 3; i < argc; i++) {
00056
00057     /* Read meteorological data... */
00058     read_met(&ctl, argv[i], met);
00059
00060     /* Find nearest pressure level... */
00061     for (ip2 = 0; ip2 < met->np; ip2++) {
00062       dz = fabs(Z(met->p[ip2]) - z);
00063       if (dz < dzmin) {
00064         dzmin = dz;
00065         ip = ip2;
00066       }
00067     }
00068
00069     /* Average data... */
00070     for (ix = 0; ix < met->nx; ix++)
00071       for (iy = 0; iy < met->ny; iy++) {
00072         intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00073                          NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL);
00074         timem[ix][iy] += met->time;
00075         zm[ix][iy] += met->z[ix][iy][ip];
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         ptm[ix][iy] += met->pt[ix][iy];
00084         ztm[ix][iy] += zt;
00085         ttm[ix][iy] += tt;
00086         np[ix][iy]++;
00087       }
00088   }
00089
00090   /* Create output file... */
00091   printf("Write meteorological data file: %s\n", argv[2]);
00092   if (!(out = fopen(argv[2], "w")))
00093     ERRMSG("Cannot create file!");
00094
00095   /* Write header... */
00096   fprintf(out,
00097           "# $1  = time [s]\n"
00098           "# $2  = altitude [km]\n"
00099           "# $3  = longitude [deg]\n"
00100           "# $4  = latitude [deg]\n"
00101           "# $5  = pressure [hPa]\n"
00102           "# $6  = temperature [K]\n"
00103           "# $7  = zonal wind [m/s]\n"
00104           "# $8  = meridional wind [m/s]\n"
00105           "# $9  = vertical wind [hPa/s]\n"
00106           "# $10 = H2O volume mixing ratio [1]\n");
00107   fprintf(out,
00108           "# $11 = O3 volume mixing ratio [1]\n"
00109           "# $12 = geopotential height [km]\n"
00110           "# $13 = surface pressure [hPa]\n"
00111           "# $14 = tropopause pressure [hPa]\n"
00112           "# $15 = tropopause geopotential height [km]\n"
00113           "# $16 = tropopause temperature [K]\n");
00114
00115   /* Write data... */
00116   for (iy = 0; iy < met->ny; iy++) {
00117     fprintf(out, "\n");
00118     for (ix = 0; ix < met->nx; ix++)
00119       if (met->lon[ix] >= 180)
00120         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00121                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00122                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00123                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00124                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00125                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00126                 zm[ix][iy] / np[ix][iy], psm[ix][iy] / np[ix][iy],
00127                 ptm[ix][iy] / np[ix][iy], ztm[ix][iy] / np[ix][iy],
00128                 ttm[ix][iy] / np[ix][iy]);
00129     for (ix = 0; ix < met->nx; ix++)
00130       if (met->lon[ix] <= 180)
00131         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00132                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
```

```
00133                 met->lon[ix], met->lat[iy], met->p[ip],
00134                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00135                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00136                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00137                 zm[ix][iy] / np[ix][iy], psm[ix][iy] / np[ix][iy],
00138                 ptm[ix][iy] / np[ix][iy], ztm[ix][iy] / np[ix][iy],
00139                 ttm[ix][iy] / np[ix][iy]);
00140   }
00141
00142   /* Close file... */
00143   fclose(out);
00144
00145   /* Free... */
00146   free(met);
00147
00148   return EXIT_SUCCESS;
00149 }
```

5.23 met_prof.c File Reference

Extract vertical profile from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.23.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met_prof.c.

5.23.2 Function Documentation

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

Definition at line 38 of file met_prof.c.

```
00040                 {
00041
00042   ctl_t ctl;
00043
00044   met_t *met;
00045
00046   FILE *out;
00047
00048   static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050     wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
00051     zg, zgm[NZ], zt, ztm[NZ];
00052
00053   static int i, iz, np[NZ];
00054
00055   /* Allocate... */
00056   ALLOC(met, met_t, 1);
00057
00058   /* Check arguments... */
00059   if (argc < 4)
00060     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00061
00062   /* Read control parameters... */
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
00065   z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
00066   dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
```

```
00067   lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00068   lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00069   dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
00070   lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
00071   lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00072   dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00073
00074   /* Loop over input files... */
00075   for (i = 3; i < argc; i++) {
00076
00077     /* Read meteorological data... */
00078     read_met(&ctl, argv[i], met);
00079
00080     /* Average... */
00081     for (z = z0; z <= z1; z += dz) {
00082       iz = (int) ((z - z0) / dz);
00083       if (iz < 0 || iz > NZ)
00084         ERRMSG("Too many altitudes!");
00085       for (lon = lon0; lon <= lon1; lon += dlon)
00086         for (lat = lat0; lat <= lat1; lat += dlat) {
00087           intpol_met_space(met, P(z), lon, lat, &ps, &pt, &zg,
00088                            &t, &u, &v, &w, &h2o, &o3);
00089           intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
00090                            &tt, NULL, NULL, NULL, NULL, NULL);
00091           if (gsl_finite(t) && gsl_finite(u)
00092               && gsl_finite(v) && gsl_finite(w)) {
00093             timem[iz] += met->time;
00094             lonm[iz] += lon;
00095             latm[iz] += lat;
00096             zgm[iz] += zg;
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            ptm[iz] += pt;
00105            ztm[iz] += zt;
00106            ttm[iz] += zt;
00107            np[iz]++;
00108          }
00109        }
00110     }
00111   }
00112
00113   /* Normalize... */
00114   for (z = z0; z <= z1; z += dz) {
00115     iz = (int) ((z - z0) / dz);
00116     if (np[iz] > 0) {
00117       timem[iz] /= np[iz];
00118       lonm[iz] /= np[iz];
00119       latm[iz] /= np[iz];
00120       zgm[iz] /= np[iz];
00121       tm[iz] /= np[iz];
00122       um[iz] /= np[iz];
00123       vm[iz] /= np[iz];
00124       wm[iz] /= np[iz];
00125       h2om[iz] /= np[iz];
00126       o3m[iz] /= np[iz];
00127       psm[iz] /= np[iz];
00128       ptm[iz] /= np[iz];
00129       ztm[iz] /= np[iz];
00130       ttm[iz] /= np[iz];
00131     } else {
00132       timem[iz] = GSL_NAN;
00133       lonm[iz] = GSL_NAN;
00134       latm[iz] = GSL_NAN;
00135       zgm[iz] = GSL_NAN;
00136       tm[iz] = GSL_NAN;
00137       um[iz] = GSL_NAN;
00138       vm[iz] = GSL_NAN;
00139       wm[iz] = GSL_NAN;
00140       h2om[iz] = GSL_NAN;
00141       o3m[iz] = GSL_NAN;
00142       psm[iz] = GSL_NAN;
00143       ptm[iz] = GSL_NAN;
00144       ztm[iz] = GSL_NAN;
00145       ttm[iz] = GSL_NAN;
00146     }
00147   }
00148
00149   /* Create output file... */
00150   printf("Write meteorological data file: %s\n", argv[2]);
00151   if (!(out = fopen(argv[2], "w")))
00152     ERRMSG("Cannot create file!");
00153
```

```
00154   /* Write header... */
00155   fprintf(out,
00156           "# $1  = time [s]\n"
00157           "# $2  = altitude [km]\n"
00158           "# $3  = longitude [deg]\n"
00159           "# $4  = latitude [deg]\n"
00160           "# $5  = pressure [hPa]\n"
00161           "# $6  = temperature [K]\n"
00162           "# $7  = zonal wind [m/s]\n"
00163           "# $8  = meridional wind [m/s]\n"
00164           "# $9  = vertical wind [hPa/s]\n");
00165   fprintf(out,
00166           "# $10 = H2O volume mixing ratio [1]\n"
00167           "# $11 = O3 volume mixing ratio [1]\n"
00168           "# $12 = geopotential height [km]\n"
00169           "# $13 = surface pressure [hPa]\n"
00170           "# $14 = tropopause pressure [hPa]\n"
00171           "# $15 = tropopause geopotential height [km]\n"
00172           "# $16 = tropopause temperature [K]\n\n");
00173
00174   /* Write data... */
00175   for (z = z0; z <= z1; z += dz) {
00176     iz = (int) ((z - z0) / dz);
00177     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00178             timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz], um[iz],
00179             vm[iz], wm[iz], h2om[iz], o3m[iz], zgm[iz], psm[iz],
00180             ptm[iz], ztm[iz], ttm[iz]);
00181   }
00182
00183   /* Close file... */
00184   fclose(out);
00185
00186   /* Free... */
00187   free(met);
00188
00189   return EXIT_SUCCESS;
00190 }
```

Here is the call graph for this function:

5.24 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 *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], w,
00050     wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
00051     zg, zgm[NZ], zt, ztm[NZ];
00052
00053   static int i, iz, np[NZ];
00054
00055   /* Allocate... */
00056   ALLOC(met, met_t, 1);
00057
00058   /* Check arguments... */
00059   if (argc < 4)
00060     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00061
00062   /* Read control parameters... */
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
00065   z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
00066   dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
00067   lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00068   lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00069   dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
00070   lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
00071   lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00072   dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00073
00074   /* Loop over input files... */
00075   for (i = 3; i < argc; i++) {
00076
00077     /* Read meteorological data... */
00078     read_met(&ctl, argv[i], met);
00079
00080     /* Average... */
00081     for (z = z0; z <= z1; z += dz) {
00082       iz = (int) ((z - z0) / dz);
00083       if (iz < 0 || iz > NZ)
00084         ERRMSG("Too many altitudes!");
00085       for (lon = lon0; lon <= lon1; lon += dlon)
00086         for (lat = lat0; lat <= lat1; lat += dlat) {
00087           intpol_met_space(met, P(z), lon, lat, &ps, &pt, &zg,
00088                            &t, &u, &v, &w, &h2o, &o3);
00089           intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
00090                            &tt, NULL, NULL, NULL, NULL, NULL);
00091           if (gsl_finite(t) && gsl_finite(u)
00092               && gsl_finite(v) && gsl_finite(w)) {
00093             timem[iz] += met->time;
00094             lonm[iz] += lon;
00095             latm[iz] += lat;
00096             zgm[iz] += zg;
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            ptm[iz] += pt;
00105            ztm[iz] += zt;
00106            ttm[iz] += zt;
00107            np[iz]++;
00108          }
00109        }
00110    }
00111  }
00112
00113  /* Normalize... */
00114  for (z = z0; z <= z1; z += dz) {
00115    iz = (int) ((z - z0) / dz);
00116    if (np[iz] > 0) {
00117      timem[iz] /= np[iz];
00118      lonm[iz] /= np[iz];
00119      latm[iz] /= np[iz];
00120      zgm[iz] /= np[iz];
00121      tm[iz] /= np[iz];
00122      um[iz] /= np[iz];
00123      vm[iz] /= np[iz];
00124      wm[iz] /= np[iz];
00125      h2om[iz] /= np[iz];
00126      o3m[iz] /= np[iz];
00127      psm[iz] /= np[iz];
00128      ptm[iz] /= np[iz];
00129      ztm[iz] /= np[iz];
00130      ttm[iz] /= np[iz];
00131    } else {
```

```
00132       timem[iz] = GSL_NAN;
00133       lonm[iz] = GSL_NAN;
00134       latm[iz] = GSL_NAN;
00135       zgm[iz] = GSL_NAN;
00136       tm[iz] = GSL_NAN;
00137       um[iz] = GSL_NAN;
00138       vm[iz] = GSL_NAN;
00139       wm[iz] = GSL_NAN;
00140       h2om[iz] = GSL_NAN;
00141       o3m[iz] = GSL_NAN;
00142       psm[iz] = GSL_NAN;
00143       ptm[iz] = GSL_NAN;
00144       ztm[iz] = GSL_NAN;
00145       ttm[iz] = GSL_NAN;
00146     }
00147   }
00148
00149   /* Create output file... */
00150   printf("Write meteorological data file: %s\n", argv[2]);
00151   if (!(out = fopen(argv[2], "w")))
00152     ERRMSG("Cannot create file!");
00153
00154   /* Write header... */
00155   fprintf(out,
00156           "# $1  = time [s]\n"
00157           "# $2  = altitude [km]\n"
00158           "# $3  = longitude [deg]\n"
00159           "# $4  = latitude [deg]\n"
00160           "# $5  = pressure [hPa]\n"
00161           "# $6  = temperature [K]\n"
00162           "# $7  = zonal wind [m/s]\n"
00163           "# $8  = meridional wind [m/s]\n"
00164           "# $9  = vertical wind [hPa/s]\n");
00165   fprintf(out,
00166           "# $10 = H2O volume mixing ratio [1]\n"
00167           "# $11 = O3 volume mixing ratio [1]\n"
00168           "# $12 = geopotential height [km]\n"
00169           "# $13 = surface pressure [hPa]\n"
00170           "# $14 = tropopause pressure [hPa]\n"
00171           "# $15 = tropopause geopotential height [km]\n"
00172           "# $16 = tropopause temperature [K]\n\n");
00173
00174   /* Write data... */
00175   for (z = z0; z <= z1; z += dz) {
00176     iz = (int) ((z - z0) / dz);
00177     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00178             timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz], um[iz],
00179             vm[iz], wm[iz], h2om[iz], o3m[iz], zgm[iz], psm[iz],
00180             ptm[iz], ztm[iz], ttm[iz]);
00181   }
00182
00183   /* Close file... */
00184   fclose(out);
00185
00186   /* Free... */
00187   free(met);
00188
00189   return EXIT_SUCCESS;
00190 }
```

5.25 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

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

5.25.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

5.25.2 Function Documentation

5.25.2.1 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 h2o, o3, ps, pt, t, tt, u, v, w, z, zt;
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   fprintf(out,
00079           "# $10 = H2O volume mixing ratio [1]\n"
00080           "# $11 = O3 volume mixing ratio [1]\n"
00081           "# $12 = geopotential height [km]\n"
00082           "# $13 = surface pressure [hPa]\n"
00083           "# $14 = tropopause pressure [hPa]\n"
00084           "# $15 = tropopause geopotential height [km]\n"
00085           "# $16 = tropopause temperature [K]\n\n");
00086
00087   /* Loop over air parcels... */
00088   for (ip = 0; ip < atm->np; ip++) {
00089
00090     /* Get meteorological data... */
00091     get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00092
00093     /* Interpolate meteorological data... */
00094     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00095                     atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &h2o, &o3);
00096     intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
      lat[ip],
00097                     NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL);
00098
00099     /* Write data... */
00100     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00101             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00102             atm->p[ip], t, u, v, w, h2o, o3, z, ps, pt, zt, tt);
00103   }
00104
00105   /* Close file... */
00106   fclose(out);
```

```
00107
00108   /* Free... */
00109   free(atm);
00110   free(met0);
00111   free(met1);
00112
00113   return EXIT_SUCCESS;
00114 }
```

Here is the call graph for this function:

5.26 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 h2o, o3, ps, pt, t, tt, u, v, w, z, zt;
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   fprintf(out,
00079           "# $10 = H2O volume mixing ratio [1]\n"
00080           "# $11 = O3 volume mixing ratio [1]\n"
00081           "# $12 = geopotential height [km]\n"
00082           "# $13 = surface pressure [hPa]\n"
00083           "# $14 = tropopause pressure [hPa]\n"
00084           "# $15 = tropopause geopotential height [km]\n"
00085           "# $16 = tropopause temperature [K]\n\n");
00086
00087   /* Loop over air parcels... */
00088   for (ip = 0; ip < atm->np; ip++) {
00089
00090     /* Get meteorological data... */
00091     get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00092
00093     /* Interpolate meteorological data... */
00094     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00095                     atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &h2o, &o3);
00096     intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
      lat[ip],
00097                     NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL);
00098
00099     /* Write data... */
00100     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00101             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00102             atm->p[ip], t, u, v, w, h2o, o3, z, ps, pt, zt, tt);
00103   }
00104
00105   /* Close file... */
00106   fclose(out);
00107
00108   /* Free... */
00109   free(atm);
00110   free(met0);
00111   free(met1);
00112
00113   return EXIT_SUCCESS;
00114 }
```

5.27 met_zm.c File Reference

Extract zonal mean from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.27.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met_zm.c.

5.27.2 Function Documentation

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

Definition at line 44 of file met_zm.c.

```
00046                {
00047
00048   ctl_t ctl;
00049
00050   met_t *met;
00051
00052   FILE *out;
00053
00054   static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], tm[EP][EY],
00055     um[EP][EY], vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY],
00056     o3m[EP][EY], zm[EP][EY], psm2[EP][EY], ptm2[EP][EY], tm2[EP][EY],
00057     um2[EP][EY], vm2[EP][EY], vhm2[EP][EY], wm2[EP][EY], h2om2[EP][EY],
00058     o3m2[EP][EY], zm2[EP][EY];
00059
00060   static int i, ip, ix, iy, np[EP][EY];
00061
00062   /* Allocate... */
00063   ALLOC(met, met_t, 1);
00064
00065   /* Check arguments... */
00066   if (argc < 4)
00067     ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00068
00069   /* Read control parameters... */
00070   read_ctl(argv[1], argc, argv, &ctl);
00071
00072   /* Loop over files... */
00073   for (i = 3; i < argc; i++) {
00074
00075     /* Read meteorological data... */
00076     read_met(&ctl, argv[i], met);
00077
00078     /* Average data... */
00079     for (ix = 0; ix < met->nx; ix++)
00080       for (iy = 0; iy < met->ny; iy++)
00081         for (ip = 0; ip < met->np; ip++) {
00082           timem[ip][iy] += met->time;
00083           psm[ip][iy] += met->ps[ix][iy];
00084           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00085           ptm[ip][iy] += met->pt[ix][iy];
00086           ptm2[ip][iy] += gsl_pow_2(met->pt[ix][iy]);
00087           zm[ip][iy] += met->z[ix][iy][ip];
00088           zm2[ip][iy] += gsl_pow_2(met->z[ix][iy][ip]);
00089           tm[ip][iy] += met->t[ix][iy][ip];
00090           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00091           um[ip][iy] += met->u[ix][iy][ip];
00092           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00093           vm[ip][iy] += met->v[ix][iy][ip];
00094           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00095           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00096                               + gsl_pow_2(met->v[ix][iy][ip]));
00097           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00098             + gsl_pow_2(met->v[ix][iy][ip]);
00099           wm[ip][iy] += met->w[ix][iy][ip];
00100           wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00101           h2om[ip][iy] += met->h2o[ix][iy][ip];
00102           h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00103           o3m[ip][iy] += met->o3[ix][iy][ip];
00104           o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00105           np[ip][iy]++;
00106         }
00107   }
00108
00109   /* Create output file... */
00110   printf("Write meteorological data file: %s\n", argv[2]);
00111   if (!(out = fopen(argv[2], "w")))
00112     ERRMSG("Cannot create file!");
00113
00114   /* Write header... */
00115   fprintf(out,
00116           "# $1  = time [s]\n"
00117           "# $2  = altitude [km]\n"
00118           "# $3  = latitude [deg]\n"
00119           "# $4  = temperature mean [K]\n"
00120           "# $5  = temperature standard deviation [K]\n"
00121           "# $6  = zonal wind mean [m/s]\n"
00122           "# $7  = zonal wind standard deviation [m/s]\n"
00123           "# $8  = meridional wind mean [m/s]\n"
00124           "# $9  = meridional wind standard deviation [m/s]\n");
00125   fprintf(out,
00126           "# $10 = horizontal wind mean [m/s]\n"
00127           "# $11 = horizontal wind standard deviation [m/s]\n"
00128           "# $12 = vertical wind mean [hPa/s]\n"
00129           "# $13 = vertical wind standard deviation [hPa/s]\n"
00130           "# $14 = H2O vmr mean [1]\n"
00131           "# $15 = H2O vmr standard deviation [1]\n"
00132           "# $16 = O3 vmr mean [1]\n"
```

```
00133           "# $17 = O3 vmr standard deviation [1]\n"
00134           "# $18 = geopotential height mean [hPa]\n"
00135           "# $19 = geopotential height standard deviation [hPa]\n");
00136   fprintf(out,
00137           "# $20 = surface pressure mean [hPa]\n"
00138           "# $21 = surface pressure standard deviation [hPa]\n"
00139           "# $22 = tropopause pressure mean [hPa]\n"
00140           "# $23 = tropopause pressure standard deviation [hPa]\n");
00141
00142   /* Write data... */
00143   for (iy = 0; iy < met->ny; iy++) {
00144     fprintf(out, "\n");
00145     for (ip = 0; ip < met->np; ip++)
00146       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00147               " %g %g %g %g %g %g %g %g %g %g %g %g\n",
00148               MEAN(timem), Z(met->p[ip]), met->lat[iy],
00149               MEAN(tm), SIGMA(tm, tm2), MEAN(um), SIGMA(um, um2),
00150               MEAN(vm), SIGMA(vm, vm2), MEAN(vhm), SIGMA(vhm, vhm2),
00151               MEAN(wm), SIGMA(wm, wm2), MEAN(h2om), SIGMA(h2om, h2om2),
00152               MEAN(o3m), SIGMA(o3m, o3m2), MEAN(zm), SIGMA(zm, zm2),
00153               MEAN(psm), SIGMA(psm, psm2), MEAN(ptm), SIGMA(ptm, ptm2));
00154   }
00155
00156   /* Close file... */
00157   fclose(out);
00158
00159   /* Free... */
00160   free(met);
00161
00162   return EXIT_SUCCESS;
00163 }
```

Here is the call graph for this function:

5.28 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 /* ------------------------------------------------------------
00028    Macros...
00029    ------------------------------------------------------------ */
00030
00032 #define MEAN(x)                                     \
00033   (x[ip][iy] / np[ip][iy])
00034
00036 #define SIGMA(x, x2)                                                    \
00037   (np[ix][iy] > 1 ? sqrt(x2[ip][iy] / np[ip][iy] -                      \
00038                          gsl_pow_2(x[ip][iy] / np[ip][iy])) : 0)
00039
00040 /* ------------------------------------------------------------
00041    Main...
00042    ------------------------------------------------------------ */
00043
00044 int main(
00045   int argc,
00046   char *argv[]) {
00047
00048   ctl_t ctl;
00049
00050   met_t *met;
00051
00052   FILE *out;
```

```
00053
00054   static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], tm[EP][EY],
00055     um[EP][EY], vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY],
00056     o3m[EP][EY], zm[EP][EY], psm2[EP][EY], ptm2[EP][EY], tm2[EP][EY],
00057     um2[EP][EY], vm2[EP][EY], vhm2[EP][EY], wm2[EP][EY], h2om2[EP][EY],
00058     o3m2[EP][EY], zm2[EP][EY];
00059
00060   static int i, ip, ix, iy, np[EP][EY];
00061
00062   /* Allocate... */
00063   ALLOC(met, met_t, 1);
00064
00065   /* Check arguments... */
00066   if (argc < 4)
00067     ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00068
00069   /* Read control parameters... */
00070   read_ctl(argv[1], argc, argv, &ctl);
00071
00072   /* Loop over files... */
00073   for (i = 3; i < argc; i++) {
00074
00075     /* Read meteorological data... */
00076     read_met(&ctl, argv[i], met);
00077
00078     /* Average data... */
00079     for (ix = 0; ix < met->nx; ix++)
00080       for (iy = 0; iy < met->ny; iy++)
00081         for (ip = 0; ip < met->np; ip++) {
00082           timem[ip][iy] += met->time;
00083           psm[ip][iy] += met->ps[ix][iy];
00084           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00085           ptm[ip][iy] += met->pt[ix][iy];
00086           ptm2[ip][iy] += gsl_pow_2(met->pt[ix][iy]);
00087           zm[ip][iy] += met->z[ix][iy][ip];
00088           zm2[ip][iy] += gsl_pow_2(met->z[ix][iy][ip]);
00089           tm[ip][iy] += met->t[ix][iy][ip];
00090           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00091           um[ip][iy] += met->u[ix][iy][ip];
00092           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00093           vm[ip][iy] += met->v[ix][iy][ip];
00094           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00095           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00096                               + gsl_pow_2(met->v[ix][iy][ip]));
00097           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00098             + gsl_pow_2(met->v[ix][iy][ip]);
00099           wm[ip][iy] += met->w[ix][iy][ip];
00100          wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00101          h2om[ip][iy] += met->h2o[ix][iy][ip];
00102          h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00103          o3m[ip][iy] += met->o3[ix][iy][ip];
00104          o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00105          np[ip][iy]++;
00106        }
00107   }
00108
00109   /* Create output file... */
00110   printf("Write meteorological data file: %s\n", argv[2]);
00111   if (!(out = fopen(argv[2], "w")))
00112     ERRMSG("Cannot create file!");
00113
00114   /* Write header... */
00115   fprintf(out,
00116           "# $1  = time [s]\n"
00117           "# $2  = altitude [km]\n"
00118           "# $3  = latitude [deg]\n"
00119           "# $4  = temperature mean [K]\n"
00120           "# $5  = temperature standard deviation [K]\n"
00121           "# $6  = zonal wind mean [m/s]\n"
00122           "# $7  = zonal wind standard deviation [m/s]\n"
00123           "# $8  = meridional wind mean [m/s]\n"
00124           "# $9  = meridional wind standard deviation [m/s]\n");
00125   fprintf(out,
00126           "# $10 = horizontal wind mean [m/s]\n"
00127           "# $11 = horizontal wind standard deviation [m/s]\n"
00128           "# $12 = vertical wind mean [hPa/s]\n"
00129           "# $13 = vertical wind standard deviation [hPa/s]\n"
00130           "# $14 = H2O vmr mean [1]\n"
00131           "# $15 = H2O vmr standard deviation [1]\n"
00132           "# $16 = O3 vmr mean [1]\n"
00133           "# $17 = O3 vmr standard deviation [1]\n"
00134           "# $18 = geopotential height mean [hPa]\n"
00135           "# $19 = geopotential height standard deviation [hPa]\n");
00136   fprintf(out,
00137           "# $20 = surface pressure mean [hPa]\n"
00138           "# $21 = surface pressure standard deviation [hPa]\n"
00139           "# $22 = tropopause pressure mean [hPa]\n"
```

```
00140           "# $23 = tropopause pressure standard deviation [hPa]\n");
00141
00142   /* Write data... */
00143   for (iy = 0; iy < met->ny; iy++) {
00144     fprintf(out, "\n");
00145     for (ip = 0; ip < met->np; ip++)
00146       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g"
00147               " %g %g %g %g %g %g %g %g %g %g %g\n",
00148               MEAN(timem), Z(met->p[ip]), met->lat[iy],
00149               MEAN(tm), SIGMA(tm, tm2), MEAN(um), SIGMA(um, um2),
00150               MEAN(vm), SIGMA(vm, vm2), MEAN(vhm), SIGMA(vhm, vhm2),
00151               MEAN(wm), SIGMA(wm, wm2), MEAN(h2om), SIGMA(h2om, h2om2),
00152               MEAN(o3m), SIGMA(o3m, o3m2), MEAN(zm), SIGMA(zm, zm2),
00153               MEAN(psm), SIGMA(psm, psm2), MEAN(ptm), SIGMA(ptm, ptm2));
00154   }
00155
00156   /* Close file... */
00157   fclose(out);
00158
00159   /* Free... */
00160   free(met);
00161
00162   return EXIT_SUCCESS;
00163 }
```

5.29 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

Functions

- int main (int argc, char ∗argv[ ])

5.29.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

5.29.2 Function Documentation

5.29.2.1 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:

5.30 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 }
```

5.31 split.c File Reference

Split air parcels into a larger number of parcels.

Functions

- int main (int argc, char ∗argv[ ])

5.31.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

5.31.2 Function Documentation

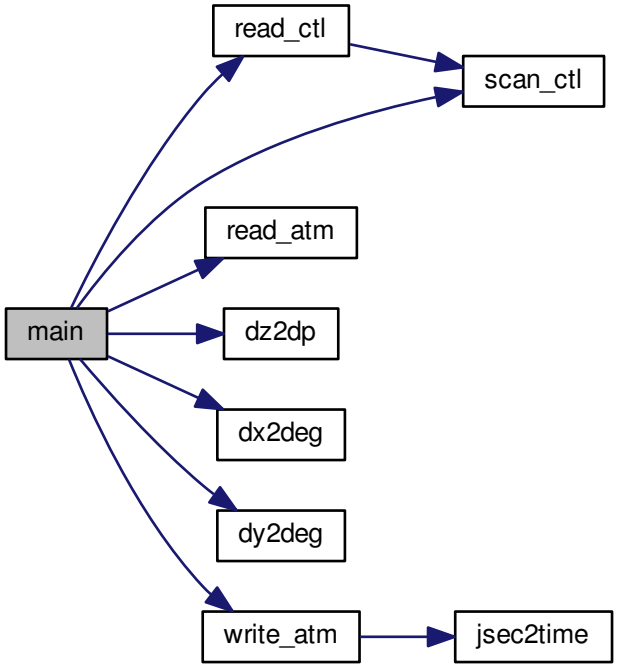
5.31.2.1 int main (int *argc*, char ∗ *argv[ ]*)

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:



5.32 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 }
```

5.33 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

- int main (int argc, char ∗argv[ ])

5.33.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

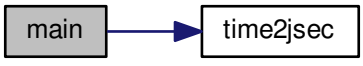
5.33.2 Function Documentation

5.33.2.1 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:



5.34 time2jsec.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 8)
00037     ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039   /* Read arguments... */
00040   year = atoi(argv[1]);
00041   mon = atoi(argv[2]);
00042   day = atoi(argv[3]);
00043   hour = atoi(argv[4]);
00044   min = atoi(argv[5]);
00045   sec = atoi(argv[6]);
00046   remain = atof(argv[7]);
00047
00048   /* Convert... */
00049   time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
00050   printf("%.2f\n", jsec);
00051
00052   return EXIT_SUCCESS;
00053 }
```

5.35 trac.c File Reference

Lagrangian particle dispersion model.

Functions

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

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

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

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

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

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

  Interpolate meteorological data for air parcel positions.

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

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

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

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

5.35.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.35.2 Function Documentation

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

Calculate advection of air parcels.

Definition at line 388 of file trac.c.

```
00393             {
00394
00395   double v[3], xm[3];
00396
00397   /* Interpolate meteorological data... */
00398   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00399                   atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00400                   &v[0], &v[1], &v[2], NULL, NULL);
00401
00402   /* Get position of the mid point... */
00403   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00404   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00405   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00406
00407   /* Interpolate meteorological data for mid point... */
00408   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00409                   xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
00410                   &v[0], &v[1], &v[2], NULL, NULL);
00411
00412   /* Save new position... */
00413   atm->time[ip] += dt;
00414   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00415   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00416   atm->p[ip] += dt * v[2];
00417 }
```

Here is the call graph for this function:

5.35.2.2 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 421 of file trac.c.

```
00427             {
00428
00429   double ps, pt, tdec;
00430
00431   /* Set constant lifetime... */
00432   if (ctl->tdec_trop == ctl->tdec_strat)
00433     tdec = ctl->tdec_trop;
00434
00435   /* Set altitude-dependent lifetime... */
00436   else {
00437
00438     /* Get surface pressure... */
00439     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00440                     atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00441                     NULL, NULL, NULL, NULL, NULL);
00442
00443     /* Get tropopause pressure... */
00444     pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00445
00446     /* Set lifetime... */
00447     if (atm->p[ip] <= pt)
00448       tdec = ctl->tdec_strat;
00449     else
00450       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00451   }
00452
00453   /* Calculate exponential decay... */
00454   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00455 }
```

Here is the call graph for this function:

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

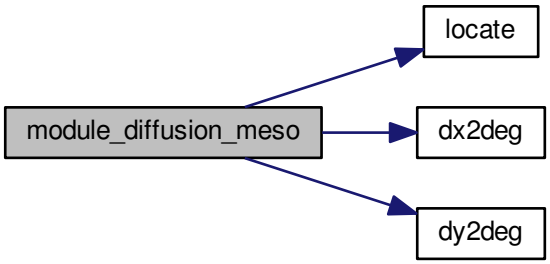
Calculate mesoscale diffusion.

Definition at line 459 of file trac.c.

```
00466                 {
00467
00468   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00469
00470   int ix, iy, iz;
00471
00472   /* Get indices... */
00473   ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00474   iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00475   iz = locate(met0->p, met0->np, atm->p[ip]);
00476
00477   /* Collect local wind data... */
00478   u[0] = met0->u[ix][iy][iz];
00479   u[1] = met0->u[ix + 1][iy][iz];
00480   u[2] = met0->u[ix][iy + 1][iz];
00481   u[3] = met0->u[ix + 1][iy + 1][iz];
00482   u[4] = met0->u[ix][iy][iz + 1];
00483   u[5] = met0->u[ix + 1][iy][iz + 1];
00484   u[6] = met0->u[ix][iy + 1][iz + 1];
00485   u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00486
00487   v[0] = met0->v[ix][iy][iz];
00488   v[1] = met0->v[ix + 1][iy][iz];
00489   v[2] = met0->v[ix][iy + 1][iz];
00490   v[3] = met0->v[ix + 1][iy + 1][iz];
00491   v[4] = met0->v[ix][iy][iz + 1];
00492   v[5] = met0->v[ix + 1][iy][iz + 1];
00493   v[6] = met0->v[ix][iy + 1][iz + 1];
00494   v[7] = met0->v[ix + 1][iy + 1][iz + 1];
```

```
00495
00496   w[0] = met0->w[ix][iy][iz];
00497   w[1] = met0->w[ix + 1][iy][iz];
00498   w[2] = met0->w[ix][iy + 1][iz];
00499   w[3] = met0->w[ix + 1][iy + 1][iz];
00500   w[4] = met0->w[ix][iy][iz + 1];
00501   w[5] = met0->w[ix + 1][iy][iz + 1];
00502   w[6] = met0->w[ix][iy + 1][iz + 1];
00503   w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00504
00505   /* Get indices... */
00506   ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00507   iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00508   iz = locate(met1->p, met1->np, atm->p[ip]);
00509
00510   /* Collect local wind data... */
00511   u[8] = met1->u[ix][iy][iz];
00512   u[9] = met1->u[ix + 1][iy][iz];
00513   u[10] = met1->u[ix][iy + 1][iz];
00514   u[11] = met1->u[ix + 1][iy + 1][iz];
00515   u[12] = met1->u[ix][iy][iz + 1];
00516   u[13] = met1->u[ix + 1][iy][iz + 1];
00517   u[14] = met1->u[ix][iy + 1][iz + 1];
00518   u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00519
00520   v[8] = met1->v[ix][iy][iz];
00521   v[9] = met1->v[ix + 1][iy][iz];
00522   v[10] = met1->v[ix][iy + 1][iz];
00523   v[11] = met1->v[ix + 1][iy + 1][iz];
00524   v[12] = met1->v[ix][iy][iz + 1];
00525   v[13] = met1->v[ix + 1][iy][iz + 1];
00526   v[14] = met1->v[ix][iy + 1][iz + 1];
00527   v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00528
00529   w[8] = met1->w[ix][iy][iz];
00530   w[9] = met1->w[ix + 1][iy][iz];
00531   w[10] = met1->w[ix][iy + 1][iz];
00532   w[11] = met1->w[ix + 1][iy + 1][iz];
00533   w[12] = met1->w[ix][iy][iz + 1];
00534   w[13] = met1->w[ix + 1][iy][iz + 1];
00535   w[14] = met1->w[ix][iy + 1][iz + 1];
00536   w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00537
00538   /* Get standard deviations of local wind data... */
00539   usig = gsl_stats_sd(u, 1, 16);
00540   vsig = gsl_stats_sd(v, 1, 16);
00541   wsig = gsl_stats_sd(w, 1, 16);
00542
00543   /* Set temporal correlations for mesoscale fluctuations... */
00544   r = 1 - 2 * fabs(dt) / ctl->dt_met;
00545   rs = sqrt(1 - r * r);
00546
00547   /* Calculate mesoscale wind fluctuations... */
00548   atm->up[ip] = (float)
00549     (r * atm->up[ip]
00550      + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * usig));
00551   atm->vp[ip] = (float)
00552     (r * atm->vp[ip]
00553      + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * vsig));
00554   atm->wp[ip] = (float)
00555     (r * atm->wp[ip]
00556      + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig));
00557
00558   /* Calculate air parcel displacement... */
00559   atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00560   atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00561   atm->p[ip] += atm->wp[ip] * dt;
00562 }
```

Here is the call graph for this function:



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

Calculate turbulent diffusion.

Definition at line 566 of file trac.c.

```
00571                 {
00572
00573   double dx, dz, pt, p0, p1, w;
00574
00575   /* Get tropopause pressure... */
00576   pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00577
00578   /* Get weighting factor... */
00579   p1 = pt * 0.866877899;
00580   p0 = pt / 0.866877899;
00581   if (atm->p[ip] > p0)
00582     w = 1;
00583   else if (atm->p[ip] < p1)
00584     w = 0;
00585   else
00586     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00587
00588   /* Set diffusivitiy... */
00589   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00590   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00591
00592   /* Horizontal turbulent diffusion... */
00593   if (dx > 0) {
00594     atm->lon[ip]
00595       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00596                 / 1000., atm->lat[ip]);
00597     atm->lat[ip]
00598       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00599                 / 1000.);
00600   }
00601
00602   /* Vertical turbulent diffusion... */
00603   if (dz > 0)
00604     atm->p[ip]
00605       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00606                / 1000., atm->p[ip]);
00607 }
```

Here is the call graph for this function:

5.35.2.5 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 611 of file trac.c.

```
00616           {
00617
00618   static double *iso, *ps, t, *ts;
00619
00620   static int idx, ip2, n;
00621
00622   FILE *in;
00623
00624   char line[LEN];
00625
00626   /* Initialize... */
00627   if (ip < 0) {
00628
00629     /* Allocate... */
00630     ALLOC(iso, double,
00631           NP);
00632     ALLOC(ps, double,
00633           NP);
00634     ALLOC(ts, double,
00635           NP);
00636
00637     /* Save pressure... */
00638     if (ctl->isosurf == 1)
00639       for (ip2 = 0; ip2 < atm->np; ip2++)
00640         iso[ip2] = atm->p[ip2];
00641
00642     /* Save density... */
00643     else if (ctl->isosurf == 2)
00644       for (ip2 = 0; ip2 < atm->np; ip2++) {
00645         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00646                         atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00647                         &t, NULL, NULL, NULL, NULL, NULL);
00648         iso[ip2] = atm->p[ip2] / t;
00649       }
00650
00651     /* Save potential temperature... */
00652     else if (ctl->isosurf == 3)
00653       for (ip2 = 0; ip2 < atm->np; ip2++) {
00654         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00655                         atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00656                         &t, NULL, NULL, NULL, NULL, NULL);
00657         iso[ip2] = THETA(atm->p[ip2], t);
00658       }
00659
00660     /* Read balloon pressure data... */
00661     else if (ctl->isosurf == 4) {
00662
00663       /* Write info... */
00664       printf("Read balloon pressure data: %s\n", ctl->balloon);
00665
00666       /* Open file... */
00667       if (!(in = fopen(ctl->balloon, "r")))
00668         ERRMSG("Cannot open file!");
00669
00670       /* Read pressure time series... */
00671       while (fgets(line, LEN, in))
00672         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00673           if ((++n) > NP)
00674             ERRMSG("Too many data points!");
00675
00676       /* Check number of points... */
00677       if (n < 1)
00678         ERRMSG("Could not read any data!");
00679
00680       /* Close file... */
00681       fclose(in);
00682     }
00683
00684     /* Leave initialization... */
00685     return;
00686   }
00687
00688   /* Restore pressure... */
00689   if (ctl->isosurf == 1)
00690     atm->p[ip] = iso[ip];
00691
00692   /* Restore density... */
```

```
00693   else if (ctl->isosurf == 2) {
00694     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00695                     atm->lat[ip], NULL, NULL, NULL, &t,
00696                     NULL, NULL, NULL, NULL, NULL);
00697     atm->p[ip] = iso[ip] * t;
00698   }
00699
00700   /* Restore potential temperature... */
00701   else if (ctl->isosurf == 3) {
00702     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00703                     atm->lat[ip], NULL, NULL, NULL, &t,
00704                     NULL, NULL, NULL, NULL, NULL);
00705     atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00706   }
00707
00708   /* Interpolate pressure... */
00709   else if (ctl->isosurf == 4) {
00710     if (atm->time[ip] <= ts[0])
00711       atm->p[ip] = ps[0];
00712     else if (atm->time[ip] >= ts[n - 1])
00713       atm->p[ip] = ps[n - 1];
00714     else {
00715       idx = locate(ts, n, atm->time[ip]);
00716       atm->p[ip] = LIN(ts[idx], ps[idx],
00717                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00718     }
00719   }
00720 }
```

Here is the call graph for this function:

5.35.2.6 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 724 of file trac.c.

```
00729           {
00730
00731   static FILE *in;
00732
00733   static char filename[2 * LEN], line[LEN];
00734
00735   static double lon[GX], lat[GY], var[GX][GY],
00736     rdum, rlat, rlat_old = -999, rlon, rvar;
00737
00738   static int year_old, mon_old, day_old, nlon, nlat;
00739
00740   double a, b, c, dp, dx, dy, dtdp, dtdx, dtdy, dudp, dudy, dvdp, dvdx,
00741     lat0, lat1, latr, lon0, lon1, ps, pt, p0, p1, p_hno3, p_h2o,
00742     t, t0, t1, u, u0, u1, v, v0, v1, w, x1, x2, h2o, o3, vort, var0, var1, z;
00743
00744   int day, mon, year, idum, ilat, ilon;
00745
00746   /* Interpolate meteorological data... */
00747   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00748                   atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &h2o, &o3);
00749
00750   /* Set surface pressure... */
00751   if (ctl->qnt_ps >= 0)
00752     atm->q[ctl->qnt_ps][ip] = ps;
00753
00754   /* Set tropopause pressure... */
00755   if (ctl->qnt_pt >= 0)
00756     atm->q[ctl->qnt_pt][ip] = pt;
00757
00758   /* Set pressure... */
00759   if (ctl->qnt_p >= 0)
00760     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00761
00762   /* Set geopotential height... */
00763   if (ctl->qnt_z >= 0)
00764     atm->q[ctl->qnt_z][ip] = z;
00765
00766   /* Set temperature... */
00767   if (ctl->qnt_t >= 0)
```

```
00768     atm->q[ctl->qnt_t][ip] = t;
00769
00770   /* Set zonal wind... */
00771   if (ctl->qnt_u >= 0)
00772     atm->q[ctl->qnt_u][ip] = u;
00773
00774   /* Set meridional wind... */
00775   if (ctl->qnt_v >= 0)
00776     atm->q[ctl->qnt_v][ip] = v;
00777
00778   /* Set vertical velocity... */
00779   if (ctl->qnt_w >= 0)
00780     atm->q[ctl->qnt_w][ip] = w;
00781
00782   /* Set water vapor vmr... */
00783   if (ctl->qnt_h2o >= 0)
00784     atm->q[ctl->qnt_h2o][ip] = h2o;
00785
00786   /* Set ozone vmr... */
00787   if (ctl->qnt_o3 >= 0)
00788     atm->q[ctl->qnt_o3][ip] = o3;
00789
00790   /* Calculate potential temperature... */
00791   if (ctl->qnt_theta >= 0)
00792     atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00793
00794   /* Calculate potential vorticity... */
00795   if (ctl->qnt_pv >= 0) {
00796
00797     /* Get gradients in longitude... */
00798     latr = GSL_MIN(GSL_MAX(atm->lat[ip], -89.), 89.);
00799     lon0 = atm->lon[ip] - (met0->lon[1] - met0->lon[0]);
00800     lon1 = atm->lon[ip] + (met0->lon[1] - met0->lon[0]);
00801     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], lon0, latr,
00802                     NULL, NULL, NULL, &t0, NULL, &v0, NULL, NULL, NULL);
00803     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], lon1, latr,
00804                     NULL, NULL, NULL, &t1, NULL, &v1, NULL, NULL, NULL);
00805     dx = 1000. * deg2dx(lon1 - lon0, latr);
00806     dtdx = (THETA(atm->p[ip], t1) - THETA(atm->p[ip], t0)) / dx;
00807     dvdx = (v1 - v0) / dx;
00808
00809     /* Get gradients in latitude... */
00810     lat0 = latr - (met0->lat[1] - met0->lat[0]);
00811     lat1 = latr + (met0->lat[1] - met0->lat[0]);
00812     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip], lat0,
00813                     NULL, NULL, NULL, &t0, &u0, NULL, NULL, NULL, NULL);
00814     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip], lat1,
00815                     NULL, NULL, NULL, &t1, &u1, NULL, NULL, NULL, NULL);
00816     dy = 1000. * deg2dy(lat1 - lat0);
00817     dtdy = (THETA(atm->p[ip], t1) - THETA(atm->p[ip], t0)) / dy;
00818     dudy = (u1 * cos(lat1 / 180. * M_PI) - u0 * cos(lat0 / 180. * M_PI)) / dy;
00819
00820     /* Get gradients in pressure... */
00821     p0 = atm->p[ip] * 0.93;
00822     p1 = atm->p[ip] / 0.93;
00823     intpol_met_time(met0, met1, atm->time[ip], p0, atm->lon[ip], latr,
00824                     NULL, NULL, NULL, &t0, &u0, &v0, NULL, NULL, NULL);
00825     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip], latr,
00826                     NULL, NULL, NULL, &t1, &u1, &v1, NULL, NULL, NULL);
00827     dp = 100. * (p1 - p0);
00828     dtdp = (THETA(p1, t1) - THETA(p0, t0)) / dp;
00829     dudp = (u1 - u0) / dp;
00830     dvdp = (v1 - v0) / dp;
00831
00832     /* Set vorticity... */
00833     vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
00834
00835     /* Calculate PV... */
00836     atm->q[ctl->qnt_pv][ip] = 1e6 * G0 *
00837       (-dtdp * (dvdx - dudy / cos(latr / 180. * M_PI) + vort) +
00838        dvdp * dtdx - dudp * dtdy);
00839   }
00840
00841   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00842   if (ctl->qnt_tice >= 0)
00843     atm->q[ctl->qnt_tice][ip] =
00844       -2663.5 /
00845       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00846        12.537);
00847
00848   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00849   if (ctl->qnt_tnat >= 0) {
00850     if (ctl->psc_hno3 > 0)
00851       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00852     else
```

```
00853       p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
00854         * 1e-9 * atm->p[ip] / 1.333224;
00855     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00856     a = 0.009179 - 0.00088 * log10(p_h2o);
00857     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00858     c = -11397.0 / a;
00859     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00860     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00861     if (x1 > 0)
00862       atm->q[ctl->qnt_tnat][ip] = x1;
00863     if (x2 > 0)
00864       atm->q[ctl->qnt_tnat][ip] = x2;
00865   }
00866
00867   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00868   if (ctl->qnt_tsts >= 0) {
00869     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00870       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00871     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00872                                        + atm->q[ctl->qnt_tnat][ip]);
00873   }
00874
00875   /* Read variance data for current day... */
00876   if (ip == 0 && ctl->qnt_gw_var >= 0) {
00877     jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00878     if (year != year_old || mon != mon_old || day != day_old) {
00879       year_old = year;
00880       mon_old = mon;
00881       day_old = day;
00882       nlon = nlat = -1;
00883       sprintf(filename, "%s_%d_%02d_%02d.tab",
00884               ctl->gw_basename, year, mon, day);
00885       if ((in = fopen(filename, "r"))) {
00886         printf("Read gravity wave data: %s\n", filename);
00887         while (fgets(line, LEN, in)) {
00888           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00889             continue;
00890           if (rlat != rlat_old) {
00891             rlat_old = rlat;
00892             if ((++nlat) > GY)
00893               ERRMSG("Too many latitudes!");
00894             nlon = -1;
00895           }
00896           if ((++nlon) > GX)
00897             ERRMSG("Too many longitudes!");
00898           lon[nlon] = rlon;
00899           lat[nlat] = rlat;
00900           var[nlon][nlat] = GSL_MAX(0, rvar);
00901         }
00902         fclose(in);
00903         nlat++;
00904         nlon++;
00905       } else
00906         printf("Warning: Missing gravity wave data: %s\n", filename);
00907     }
00908   }
00909
00910   /* Interpolate variance data... */
00911   if (ctl->qnt_gw_var >= 0) {
00912     if (nlat >= 2 && nlon >= 2) {
00913       ilat = locate(lat, nlat, atm->lat[ip]);
00914       ilon = locate(lon, nlon, atm->lon[ip]);
00915       var0 = LIN(lat[ilat], var[ilon][ilat],
00916                  lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00917       var1 = LIN(lat[ilat], var[ilon + 1][ilat],
00918                  lat[ilat + 1], var[ilon + 1][ilat + 1], atm->lat[ip]);
00919       atm->q[ctl->qnt_gw_var][ip]
00920         = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00921     } else
00922       atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00923   }
00924 }
```

Here is the call graph for this function:

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

Check position of air parcels.

Definition at line 928 of file trac.c.

```
00932           {
00933
00934     double ps;
00935
00936     /* Calculate modulo... */
00937     atm->lon[ip] = fmod(atm->lon[ip], 360);
00938     atm->lat[ip] = fmod(atm->lat[ip], 360);
00939
00940     /* Check latitude... */
00941     while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00942       if (atm->lat[ip] > 90) {
00943         atm->lat[ip] = 180 - atm->lat[ip];
00944         atm->lon[ip] += 180;
00945       }
00946       if (atm->lat[ip] < -90) {
00947         atm->lat[ip] = -180 - atm->lat[ip];
00948         atm->lon[ip] += 180;
00949       }
00950     }
00951
00952     /* Check longitude... */
00953     while (atm->lon[ip] < -180)
00954       atm->lon[ip] += 360;
00955     while (atm->lon[ip] >= 180)
00956       atm->lon[ip] -= 360;
00957
00958     /* Get surface pressure... */
00959     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00960                     atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00961                     NULL, NULL, NULL, NULL, NULL);
00962
00963     /* Check pressure... */
00964     if (atm->p[ip] > ps)
00965       atm->p[ip] = ps;
00966     else if (atm->p[ip] < met0->p[met0->np - 1])
00967       atm->p[ip] = met0->p[met0->np - 1];
00968   }
```

Here is the call graph for this function:

5.35.2.8 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 972 of file trac.c.

```
00978             {
00979
00980     /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00981     const double A = 1.249, B = 0.42, C = 0.87;
00982
00983     /* Average mass of an air molecule [kg/molec]: */
00984     const double m = 4.8096e-26;
00985
00986     double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00987
00988     /* Convert units... */
00989     p = 100 * atm->p[ip];
00990     r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00991     rho_p = atm->q[ctl->qnt_rho][ip];
00992
00993     /* Get temperature... */
00994     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
        lon[ip],
00995                     atm->lat[ip], NULL, NULL, NULL, &T,
00996                     NULL, NULL, NULL, NULL, NULL);
00997
00998     /* Density of dry air... */
00999     rho = p / (R0 * T);
01000
01001     /* Dynamic viscosity of air... */
01002     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01003
01004     /* Thermal velocity of an air molecule... */
01005     v = sqrt(8 * KB * T / (M_PI * m));
01006
01007     /* Mean free path of an air molecule... */
01008     lambda = 2 * eta / (rho * v);
01009
```

```
01010   /* Knudsen number for air... */
01011   K = lambda / r_p;
01012
01013   /* Cunningham slip-flow correction... */
01014   G = 1 + K * (A + B * exp(-C / K));
01015
01016   /* Sedimentation (fall) velocity... */
01017   v_p = 2. * gsl_pow_2(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01018
01019   /* Calculate pressure change... */
01020   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01021 }
```

Here is the call graph for this function:

5.35.2.9 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 1025 of file trac.c.

```
01031             {
01032
01033   char filename[2 * LEN];
01034
01035   double r;
01036
01037   int year, mon, day, hour, min, sec;
01038
01039   /* Get time... */
01040   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01041
01042   /* Write atmospheric data... */
01043   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01044     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01045             dirname, ctl->atm_basename, year, mon, day, hour, min);
01046     write_atm(filename, ctl, atm, t);
01047   }
01048
01049   /* Write CSI data... */
01050   if (ctl->csi_basename[0] != '-') {
01051     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01052     write_csi(filename, ctl, atm, t);
01053   }
01054
01055   /* Write ensemble data... */
01056   if (ctl->ens_basename[0] != '-') {
01057     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01058     write_ens(filename, ctl, atm, t);
01059   }
01060
01061   /* Write gridded data... */
01062   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01063     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01064             dirname, ctl->grid_basename, year, mon, day, hour, min);
01065     write_grid(filename, ctl, met0, met1, atm, t);
01066   }
01067
01068   /* Write profile data... */
01069   if (ctl->prof_basename[0] != '-') {
01070     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01071     write_prof(filename, ctl, met0, met1, atm, t);
01072   }
01073
01074   /* Write station data... */
01075   if (ctl->stat_basename[0] != '-') {
01076     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01077     write_station(filename, ctl, atm, t);
01078   }
01079 }
```

Here is the call graph for this function:

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

Definition at line 115 of file trac.c.

```
00117                {
00118
00119   ctl_t ctl;
00120
00121   atm_t *atm;
00122
00123   met_t *met0, *met1;
00124
00125   gsl_rng *rng[NTHREADS];
00126
00127   FILE *dirlist;
00128
00129   char dirname[LEN], filename[LEN];
00130
00131   double *dt, t, t0;
00132
00133   int i, ip, ntask = 0, rank = 0, size = 1;
00134
00135 #ifdef MPI
00136   /* Initialize MPI... */
00137   MPI_Init(&argc, &argv);
00138   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00139   MPI_Comm_size(MPI_COMM_WORLD, &size);
00140 #endif
00141
00142   /* Check arguments... */
00143   if (argc < 5)
00144     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00145
00146   /* Open directory list... */
00147   if (!(dirlist = fopen(argv[1], "r")))
00148     ERRMSG("Cannot open directory list!");
00149
00150   /* Loop over directories... */
00151   while (fscanf(dirlist, "%s", dirname) != EOF) {
00152
00153     /* MPI parallelization... */
00154     if ((++ntask) % size != rank)
00155       continue;
00156
00157     /* ------------------------------------------------------------
00158        Initialize model run...
00159        ------------------------------------------------------------ */
00160
00161     /* Set timers... */
00162     START_TIMER(TIMER_TOTAL);
00163     START_TIMER(TIMER_INIT);
00164
00165     /* Allocate... */
00166     ALLOC(atm, atm_t, 1);
00167     ALLOC(met0, met_t, 1);
00168     ALLOC(met1, met_t, 1);
00169     ALLOC(dt, double,
00170           NP);
00171
00172     /* Initialize random number generators... */
00173     gsl_rng_env_setup();
00174     if (omp_get_max_threads() > NTHREADS)
00175       ERRMSG("Too many threads!");
00176     for (i = 0; i < NTHREADS; i++) {
00177       rng[i] = gsl_rng_alloc(gsl_rng_default);
00178       gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00179     }
00180
00181     /* Read control parameters... */
00182     sprintf(filename, "%s/%s", dirname, argv[2]);
00183     read_ctl(filename, argc, argv, &ctl);
00184
00185     /* Read atmospheric data... */
00186     sprintf(filename, "%s/%s", dirname, argv[3]);
00187     read_atm(filename, &ctl, atm);
00188
00189     /* Set inital and final time... */
00190     if (ctl.direction == 1) {
00191       if (ctl.t_start < -1e99)
00192         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00193       if (ctl.t_stop < -1e99)
00194         ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00195     } else if (ctl.direction == -1) {
00196       if (ctl.t_stop < -1e99)
```

```
00197         ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00198       if (ctl.t_start < -1e99)
00199         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00200     }
00201
00202     /* Check time... */
00203     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00204       ERRMSG("Nothing to do!");
00205
00206     /* Get rounded start time... */
00207     if (ctl.direction == 1)
00208       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00209     else
00210       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00211
00212     /* Set timers... */
00213     STOP_TIMER(TIMER_INIT);
00214
00215     /* ------------------------------------------------------------
00216        Loop over timesteps...
00217        ------------------------------------------------------------ */
00218
00219     /* Loop over timesteps... */
00220     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00221          t += ctl.direction * ctl.dt_mod) {
00222
00223       /* Adjust length of final time step... */
00224       if (ctl.direction * (t - ctl.t_stop) > 0)
00225         t = ctl.t_stop;
00226
00227       /* Set time steps for air parcels... */
00228       for (ip = 0; ip < atm->np; ip++)
00229         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00230              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00231              && ctl.direction * (atm->time[ip] - t) < 0))
00232           dt[ip] = t - atm->time[ip];
00233         else
00234           dt[ip] = GSL_NAN;
00235
00236       /* Get meteorological data... */
00237       START_TIMER(TIMER_INPUT);
00238       get_met(&ctl, argv[4], t, met0, met1);
00239       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00240         printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00241                fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00242       STOP_TIMER(TIMER_INPUT);
00243
00244       /* Initialize isosurface... */
00245       START_TIMER(TIMER_ISOSURF);
00246       if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00247         if (t == t0)
00248           module_isosurf(&ctl, met0, met1, atm, -1);
00249       STOP_TIMER(TIMER_ISOSURF);
00250
00251       /* Advection... */
00252       START_TIMER(TIMER_ADVECT);
00253 #pragma omp parallel for default(shared) private(ip)
00254       for (ip = 0; ip < atm->np; ip++)
00255         if (gsl_finite(dt[ip]))
00256           module_advection(met0, met1, atm, ip, dt[ip]);
00257       STOP_TIMER(TIMER_ADVECT);
00258
00259       /* Turbulent diffusion... */
00260       START_TIMER(TIMER_DIFFTURB);
00261       if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00262           || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00263 #pragma omp parallel for default(shared) private(ip)
00264         for (ip = 0; ip < atm->np; ip++)
00265           if (gsl_finite(dt[ip]))
00266             module_diffusion_turb(&ctl, atm, ip, dt[ip],
00267                                   rng[omp_get_thread_num()]);
00268       }
00269       STOP_TIMER(TIMER_DIFFTURB);
00270
00271       /* Mesoscale diffusion... */
00272       START_TIMER(TIMER_DIFFMESO);
00273       if (ctl.turb_meso > 0) {
00274 #pragma omp parallel for default(shared) private(ip)
00275         for (ip = 0; ip < atm->np; ip++)
00276           if (gsl_finite(dt[ip]))
00277             module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00278                                   rng[omp_get_thread_num()]);
00279       }
00280       STOP_TIMER(TIMER_DIFFMESO);
00281
00282       /* Sedimentation... */
00283       START_TIMER(TIMER_SEDI);
```

```
00284       if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
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_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00289       }
00290       STOP_TIMER(TIMER_SEDI);
00291
00292       /* Isosurface... */
00293       START_TIMER(TIMER_ISOSURF);
00294       if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
00295 #pragma omp parallel for default(shared) private(ip)
00296         for (ip = 0; ip < atm->np; ip++)
00297           module_isosurf(&ctl, met0, met1, atm, ip);
00298       }
00299       STOP_TIMER(TIMER_ISOSURF);
00300
00301       /* Position... */
00302       START_TIMER(TIMER_POSITION);
00303 #pragma omp parallel for default(shared) private(ip)
00304       for (ip = 0; ip < atm->np; ip++)
00305         module_position(met0, met1, atm, ip);
00306       STOP_TIMER(TIMER_POSITION);
00307
00308       /* Meteorological data... */
00309       START_TIMER(TIMER_METEO);
00310       module_meteo(&ctl, met0, met1, atm, 0);
00311 #pragma omp parallel for default(shared) private(ip)
00312       for (ip = 1; ip < atm->np; ip++)
00313         module_meteo(&ctl, met0, met1, atm, ip);
00314       STOP_TIMER(TIMER_METEO);
00315
00316       /* Decay... */
00317       START_TIMER(TIMER_DECAY);
00318       if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.
      qnt_m >= 0) {
00319 #pragma omp parallel for default(shared) private(ip)
00320         for (ip = 0; ip < atm->np; ip++)
00321           if (gsl_finite(dt[ip]))
00322             module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00323       }
00324       STOP_TIMER(TIMER_DECAY);
00325
00326       /* Write output... */
00327       START_TIMER(TIMER_OUTPUT);
00328       write_output(dirname, &ctl, met0, met1, atm, t);
00329       STOP_TIMER(TIMER_OUTPUT);
00330     }
00331
00332     /* ------------------------------------------------------------
00333        Finalize model run...
00334        ------------------------------------------------------------ */
00335
00336     /* Report memory usage... */
00337     printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00338     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00339     printf("MEMORY_DYNAMIC = %g MByte\n",
00340            4 * NP * sizeof(double) / 1024. / 1024.);
00341     printf("MEMORY_STATIC = %g MByte\n",
00342            ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
00343             + (EX * EY + EX * EY * EP) * sizeof(float)
00344             + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00345
00346     /* Report problem size... */
00347     printf("SIZE_NP = %d\n", atm->np);
00348     printf("SIZE_TASKS = %d\n", size);
00349     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00350
00351     /* Report timers... */
00352     STOP_TIMER(TIMER_TOTAL);
00353     PRINT_TIMER(TIMER_TOTAL);
00354     PRINT_TIMER(TIMER_INIT);
00355     PRINT_TIMER(TIMER_STAGE);
00356     PRINT_TIMER(TIMER_INPUT);
00357     PRINT_TIMER(TIMER_OUTPUT);
00358     PRINT_TIMER(TIMER_ADVECT);
00359     PRINT_TIMER(TIMER_DECAY);
00360     PRINT_TIMER(TIMER_DIFFMESO);
00361     PRINT_TIMER(TIMER_DIFFTURB);
00362     PRINT_TIMER(TIMER_ISOSURF);
00363     PRINT_TIMER(TIMER_METEO);
00364     PRINT_TIMER(TIMER_POSITION);
00365     PRINT_TIMER(TIMER_SEDI);
00366
00367     /* Free random number generators... */
00368     for (i = 0; i < NTHREADS; i++)
00369       gsl_rng_free(rng[i]);
```

```
00370
00371     /* Free... */
00372     free(atm);
00373     free(met0);
00374     free(met1);
00375     free(dt);
00376   }
00377
00378 #ifdef MPI
00379   /* Finalize MPI... */
00380   MPI_Finalize();
00381 #endif
00382
00383   return EXIT_SUCCESS;
00384 }
```

Here is the call graph for this function:

5.36 trac.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
00030
00031 /* ------------------------------------------------------------
00032    Functions...
00033    ------------------------------------------------------------ */
00034
00036 void module_advection(
00037   met_t * met0,
00038   met_t * met1,
00039   atm_t * atm,
00040   int ip,
00041   double dt);
00042
00044 void module_decay(
00045   ctl_t * ctl,
00046   met_t * met0,
00047   met_t * met1,
00048   atm_t * atm,
00049   int ip,
00050   double dt);
00051
00053 void module_diffusion_meso(
00054   ctl_t * ctl,
00055   met_t * met0,
00056   met_t * met1,
00057   atm_t * atm,
00058   int ip,
00059   double dt,
00060   gsl_rng * rng);
00061
00063 void module_diffusion_turb(
00064   ctl_t * ctl,
00065   atm_t * atm,
00066   int ip,
00067   double dt,
00068   gsl_rng * rng);
00069
00071 void module_isosurf(
```

```
00072   ctl_t * ctl,
00073   met_t * met0,
00074   met_t * met1,
00075   atm_t * atm,
00076   int ip);
00077
00079 void module_meteo(
00080   ctl_t * ctl,
00081   met_t * met0,
00082   met_t * met1,
00083   atm_t * atm,
00084   int ip);
00085
00087 void module_position(
00088   met_t * met0,
00089   met_t * met1,
00090   atm_t * atm,
00091   int ip);
00092
00094 void module_sedi(
00095   ctl_t * ctl,
00096   met_t * met0,
00097   met_t * met1,
00098   atm_t * atm,
00099   int ip,
00100   double dt);
00101
00103 void write_output(
00104   const char *dirname,
00105   ctl_t * ctl,
00106   met_t * met0,
00107   met_t * met1,
00108   atm_t * atm,
00109   double t);
00110
00111 /* ------------------------------------------------------------
00112    Main...
00113    ------------------------------------------------------------ */
00114
00115 int main(
00116   int argc,
00117   char *argv[]) {
00118
00119   ctl_t ctl;
00120
00121   atm_t *atm;
00122
00123   met_t *met0, *met1;
00124
00125   gsl_rng *rng[NTHREADS];
00126
00127   FILE *dirlist;
00128
00129   char dirname[LEN], filename[LEN];
00130
00131   double *dt, t, t0;
00132
00133   int i, ip, ntask = 0, rank = 0, size = 1;
00134
00135 #ifdef MPI
00136   /* Initialize MPI... */
00137   MPI_Init(&argc, &argv);
00138   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00139   MPI_Comm_size(MPI_COMM_WORLD, &size);
00140 #endif
00141
00142   /* Check arguments... */
00143   if (argc < 5)
00144     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00145
00146   /* Open directory list... */
00147   if (!(dirlist = fopen(argv[1], "r")))
00148     ERRMSG("Cannot open directory list!");
00149
00150   /* Loop over directories... */
00151   while (fscanf(dirlist, "%s", dirname) != EOF) {
00152
00153     /* MPI parallelization... */
00154     if ((++ntask) % size != rank)
00155       continue;
00156
00157     /* ------------------------------------------------------------
00158        Initialize model run...
00159        ------------------------------------------------------------ */
00160
00161     /* Set timers... */
00162     START_TIMER(TIMER_TOTAL);
```

```
00163     START_TIMER(TIMER_INIT);
00164
00165     /* Allocate... */
00166     ALLOC(atm, atm_t, 1);
00167     ALLOC(met0, met_t, 1);
00168     ALLOC(met1, met_t, 1);
00169     ALLOC(dt, double,
00170           NP);
00171
00172     /* Initialize random number generators... */
00173     gsl_rng_env_setup();
00174     if (omp_get_max_threads() > NTHREADS)
00175       ERRMSG("Too many threads!");
00176     for (i = 0; i < NTHREADS; i++) {
00177       rng[i] = gsl_rng_alloc(gsl_rng_default);
00178       gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00179     }
00180
00181     /* Read control parameters... */
00182     sprintf(filename, "%s/%s", dirname, argv[2]);
00183     read_ctl(filename, argc, argv, &ctl);
00184
00185     /* Read atmospheric data... */
00186     sprintf(filename, "%s/%s", dirname, argv[3]);
00187     read_atm(filename, &ctl, atm);
00188
00189     /* Set inital and final time... */
00190     if (ctl.direction == 1) {
00191       if (ctl.t_start < -1e99)
00192         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00193       if (ctl.t_stop < -1e99)
00194         ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00195     } else if (ctl.direction == -1) {
00196       if (ctl.t_stop < -1e99)
00197         ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00198       if (ctl.t_start < -1e99)
00199         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00200     }
00201
00202     /* Check time... */
00203     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00204       ERRMSG("Nothing to do!");
00205
00206     /* Get rounded start time... */
00207     if (ctl.direction == 1)
00208       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00209     else
00210       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00211
00212     /* Set timers... */
00213     STOP_TIMER(TIMER_INIT);
00214
00215     /* ------------------------------------------------------------
00216        Loop over timesteps...
00217        ------------------------------------------------------------ */
00218
00219     /* Loop over timesteps... */
00220     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00221          t += ctl.direction * ctl.dt_mod) {
00222
00223       /* Adjust length of final time step... */
00224       if (ctl.direction * (t - ctl.t_stop) > 0)
00225         t = ctl.t_stop;
00226
00227       /* Set time steps for air parcels... */
00228       for (ip = 0; ip < atm->np; ip++)
00229         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00230              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00231              && ctl.direction * (atm->time[ip] - t) < 0))
00232           dt[ip] = t - atm->time[ip];
00233         else
00234           dt[ip] = GSL_NAN;
00235
00236       /* Get meteorological data... */
00237       START_TIMER(TIMER_INPUT);
00238       get_met(&ctl, argv[4], t, met0, met1);
00239       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00240         printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00241                fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00242       STOP_TIMER(TIMER_INPUT);
00243
00244       /* Initialize isosurface... */
00245       START_TIMER(TIMER_ISOSURF);
00246       if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00247         if (t == t0)
00248           module_isosurf(&ctl, met0, met1, atm, -1);
00249       STOP_TIMER(TIMER_ISOSURF);
```

```
00250
00251       /* Advection... */
00252       START_TIMER(TIMER_ADVECT);
00253 #pragma omp parallel for default(shared) private(ip)
00254       for (ip = 0; ip < atm->np; ip++)
00255         if (gsl_finite(dt[ip]))
00256           module_advection(met0, met1, atm, ip, dt[ip]);
00257       STOP_TIMER(TIMER_ADVECT);
00258
00259       /* Turbulent diffusion... */
00260       START_TIMER(TIMER_DIFFTURB);
00261       if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00262           || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00263 #pragma omp parallel for default(shared) private(ip)
00264         for (ip = 0; ip < atm->np; ip++)
00265           if (gsl_finite(dt[ip]))
00266             module_diffusion_turb(&ctl, atm, ip, dt[ip],
00267                                   rng[omp_get_thread_num()]);
00268       }
00269       STOP_TIMER(TIMER_DIFFTURB);
00270
00271       /* Mesoscale diffusion... */
00272       START_TIMER(TIMER_DIFFMESO);
00273       if (ctl.turb_meso > 0) {
00274 #pragma omp parallel for default(shared) private(ip)
00275         for (ip = 0; ip < atm->np; ip++)
00276           if (gsl_finite(dt[ip]))
00277             module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00278                                   rng[omp_get_thread_num()]);
00279       }
00280       STOP_TIMER(TIMER_DIFFMESO);
00281
00282       /* Sedimentation... */
00283       START_TIMER(TIMER_SEDI);
00284       if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
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_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00289       }
00290       STOP_TIMER(TIMER_SEDI);
00291
00292       /* Isosurface... */
00293       START_TIMER(TIMER_ISOSURF);
00294       if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
00295 #pragma omp parallel for default(shared) private(ip)
00296         for (ip = 0; ip < atm->np; ip++)
00297           module_isosurf(&ctl, met0, met1, atm, ip);
00298       }
00299       STOP_TIMER(TIMER_ISOSURF);
00300
00301       /* Position... */
00302       START_TIMER(TIMER_POSITION);
00303 #pragma omp parallel for default(shared) private(ip)
00304       for (ip = 0; ip < atm->np; ip++)
00305         module_position(met0, met1, atm, ip);
00306       STOP_TIMER(TIMER_POSITION);
00307
00308       /* Meteorological data... */
00309       START_TIMER(TIMER_METEO);
00310       module_meteo(&ctl, met0, met1, atm, 0);
00311 #pragma omp parallel for default(shared) private(ip)
00312       for (ip = 1; ip < atm->np; ip++)
00313         module_meteo(&ctl, met0, met1, atm, ip);
00314       STOP_TIMER(TIMER_METEO);
00315
00316       /* Decay... */
00317       START_TIMER(TIMER_DECAY);
00318       if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.
      qnt_m >= 0) {
00319 #pragma omp parallel for default(shared) private(ip)
00320         for (ip = 0; ip < atm->np; ip++)
00321           if (gsl_finite(dt[ip]))
00322             module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00323       }
00324       STOP_TIMER(TIMER_DECAY);
00325
00326       /* Write output... */
00327       START_TIMER(TIMER_OUTPUT);
00328       write_output(dirname, &ctl, met0, met1, atm, t);
00329       STOP_TIMER(TIMER_OUTPUT);
00330     }
00331
00332     /* ------------------------------------------------------------
00333        Finalize model run...
00334        ------------------------------------------------------------ */
00335
```

```
00336     /* Report memory usage... */
00337     printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00338     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00339     printf("MEMORY_DYNAMIC = %g MByte\n",
00340            4 * NP * sizeof(double) / 1024. / 1024.);
00341     printf("MEMORY_STATIC = %g MByte\n",
00342            ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
00343             + (EX * EY + EX * EY * EP) * sizeof(float)
00344             + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00345
00346     /* Report problem size... */
00347     printf("SIZE_NP = %d\n", atm->np);
00348     printf("SIZE_TASKS = %d\n", size);
00349     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00350
00351     /* Report timers... */
00352     STOP_TIMER(TIMER_TOTAL);
00353     PRINT_TIMER(TIMER_TOTAL);
00354     PRINT_TIMER(TIMER_INIT);
00355     PRINT_TIMER(TIMER_STAGE);
00356     PRINT_TIMER(TIMER_INPUT);
00357     PRINT_TIMER(TIMER_OUTPUT);
00358     PRINT_TIMER(TIMER_ADVECT);
00359     PRINT_TIMER(TIMER_DECAY);
00360     PRINT_TIMER(TIMER_DIFFMESO);
00361     PRINT_TIMER(TIMER_DIFFTURB);
00362     PRINT_TIMER(TIMER_ISOSURF);
00363     PRINT_TIMER(TIMER_METEO);
00364     PRINT_TIMER(TIMER_POSITION);
00365     PRINT_TIMER(TIMER_SEDI);
00366
00367     /* Free random number generators... */
00368     for (i = 0; i < NTHREADS; i++)
00369       gsl_rng_free(rng[i]);
00370
00371     /* Free... */
00372     free(atm);
00373     free(met0);
00374     free(met1);
00375     free(dt);
00376   }
00377
00378 #ifdef MPI
00379   /* Finalize MPI... */
00380   MPI_Finalize();
00381 #endif
00382
00383   return EXIT_SUCCESS;
00384 }
00385
00386 /*****************************************************************************/
00387
00388 void module_advection(
00389   met_t * met0,
00390   met_t * met1,
00391   atm_t * atm,
00392   int ip,
00393   double dt) {
00394
00395   double v[3], xm[3];
00396
00397   /* Interpolate meteorological data... */
00398   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00399                   atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00400                   &v[0], &v[1], &v[2], NULL, NULL);
00401
00402   /* Get position of the mid point... */
00403   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00404   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00405   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00406
00407   /* Interpolate meteorological data for mid point... */
00408   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00409                   xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
00410                   &v[0], &v[1], &v[2], NULL, NULL);
00411
00412   /* Save new position... */
00413   atm->time[ip] += dt;
00414   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00415   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00416   atm->p[ip] += dt * v[2];
00417 }
00418
00419 /*****************************************************************************/
00420
00421 void module_decay(
00422   ctl_t * ctl,
```

```
00423   met_t * met0,
00424   met_t * met1,
00425   atm_t * atm,
00426   int ip,
00427   double dt) {
00428
00429   double ps, pt, tdec;
00430
00431   /* Set constant lifetime... */
00432   if (ctl->tdec_trop == ctl->tdec_strat)
00433     tdec = ctl->tdec_trop;
00434
00435   /* Set altitude-dependent lifetime... */
00436   else {
00437
00438     /* Get surface pressure... */
00439     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00440                     atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00441                     NULL, NULL, NULL, NULL, NULL);
00442
00443     /* Get tropopause pressure... */
00444     pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00445
00446     /* Set lifetime... */
00447     if (atm->p[ip] <= pt)
00448       tdec = ctl->tdec_strat;
00449     else
00450       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00451   }
00452
00453   /* Calculate exponential decay... */
00454   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00455 }
00456
00457 /*****************************************************************************/
00458
00459 void module_diffusion_meso(
00460   ctl_t * ctl,
00461   met_t * met0,
00462   met_t * met1,
00463   atm_t * atm,
00464   int ip,
00465   double dt,
00466   gsl_rng * rng) {
00467
00468   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00469
00470   int ix, iy, iz;
00471
00472   /* Get indices... */
00473   ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00474   iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00475   iz = locate(met0->p, met0->np, atm->p[ip]);
00476
00477   /* Collect local wind data... */
00478   u[0] = met0->u[ix][iy][iz];
00479   u[1] = met0->u[ix + 1][iy][iz];
00480   u[2] = met0->u[ix][iy + 1][iz];
00481   u[3] = met0->u[ix + 1][iy + 1][iz];
00482   u[4] = met0->u[ix][iy][iz + 1];
00483   u[5] = met0->u[ix + 1][iy][iz + 1];
00484   u[6] = met0->u[ix][iy + 1][iz + 1];
00485   u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00486
00487   v[0] = met0->v[ix][iy][iz];
00488   v[1] = met0->v[ix + 1][iy][iz];
00489   v[2] = met0->v[ix][iy + 1][iz];
00490   v[3] = met0->v[ix + 1][iy + 1][iz];
00491   v[4] = met0->v[ix][iy][iz + 1];
00492   v[5] = met0->v[ix + 1][iy][iz + 1];
00493   v[6] = met0->v[ix][iy + 1][iz + 1];
00494   v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00495
00496   w[0] = met0->w[ix][iy][iz];
00497   w[1] = met0->w[ix + 1][iy][iz];
00498   w[2] = met0->w[ix][iy + 1][iz];
00499   w[3] = met0->w[ix + 1][iy + 1][iz];
00500   w[4] = met0->w[ix][iy][iz + 1];
00501   w[5] = met0->w[ix + 1][iy][iz + 1];
00502   w[6] = met0->w[ix][iy + 1][iz + 1];
00503   w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00504
00505   /* Get indices... */
00506   ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00507   iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00508   iz = locate(met1->p, met1->np, atm->p[ip]);
```

```
00509
00510   /* Collect local wind data... */
00511   u[8] = met1->u[ix][iy][iz];
00512   u[9] = met1->u[ix + 1][iy][iz];
00513   u[10] = met1->u[ix][iy + 1][iz];
00514   u[11] = met1->u[ix + 1][iy + 1][iz];
00515   u[12] = met1->u[ix][iy][iz + 1];
00516   u[13] = met1->u[ix + 1][iy][iz + 1];
00517   u[14] = met1->u[ix][iy + 1][iz + 1];
00518   u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00519
00520   v[8] = met1->v[ix][iy][iz];
00521   v[9] = met1->v[ix + 1][iy][iz];
00522   v[10] = met1->v[ix][iy + 1][iz];
00523   v[11] = met1->v[ix + 1][iy + 1][iz];
00524   v[12] = met1->v[ix][iy][iz + 1];
00525   v[13] = met1->v[ix + 1][iy][iz + 1];
00526   v[14] = met1->v[ix][iy + 1][iz + 1];
00527   v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00528
00529   w[8] = met1->w[ix][iy][iz];
00530   w[9] = met1->w[ix + 1][iy][iz];
00531   w[10] = met1->w[ix][iy + 1][iz];
00532   w[11] = met1->w[ix + 1][iy + 1][iz];
00533   w[12] = met1->w[ix][iy][iz + 1];
00534   w[13] = met1->w[ix + 1][iy][iz + 1];
00535   w[14] = met1->w[ix][iy + 1][iz + 1];
00536   w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00537
00538   /* Get standard deviations of local wind data... */
00539   usig = gsl_stats_sd(u, 1, 16);
00540   vsig = gsl_stats_sd(v, 1, 16);
00541   wsig = gsl_stats_sd(w, 1, 16);
00542
00543   /* Set temporal correlations for mesoscale fluctuations... */
00544   r = 1 - 2 * fabs(dt) / ctl->dt_met;
00545   rs = sqrt(1 - r * r);
00546
00547   /* Calculate mesoscale wind fluctuations... */
00548   atm->up[ip] = (float)
00549     (r * atm->up[ip]
00550      + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * usig));
00551   atm->vp[ip] = (float)
00552     (r * atm->vp[ip]
00553      + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * vsig));
00554   atm->wp[ip] = (float)
00555     (r * atm->wp[ip]
00556      + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig));
00557
00558   /* Calculate air parcel displacement... */
00559   atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00560   atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00561   atm->p[ip] += atm->wp[ip] * dt;
00562 }
00563
00564 /*****************************************************************************/
00565
00566 void module_diffusion_turb(
00567   ctl_t * ctl,
00568   atm_t * atm,
00569   int ip,
00570   double dt,
00571   gsl_rng * rng) {
00572
00573   double dx, dz, pt, p0, p1, w;
00574
00575   /* Get tropopause pressure... */
00576   pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00577
00578   /* Get weighting factor... */
00579   p1 = pt * 0.866877899;
00580   p0 = pt / 0.866877899;
00581   if (atm->p[ip] > p0)
00582     w = 1;
00583   else if (atm->p[ip] < p1)
00584     w = 0;
00585   else
00586     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00587
00588   /* Set diffusivitiy... */
00589   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00590   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00591
00592   /* Horizontal turbulent diffusion... */
00593   if (dx > 0) {
00594     atm->lon[ip]
00595       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
```

```
00596                 / 1000., atm->lat[ip]);
00597     atm->lat[ip]
00598       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00599                 / 1000.);
00600   }
00601
00602   /* Vertical turbulent diffusion... */
00603   if (dz > 0)
00604     atm->p[ip]
00605       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00606                / 1000., atm->p[ip]);
00607 }
00608
00609 /*****************************************************************************/
00610
00611 void module_isosurf(
00612   ctl_t * ctl,
00613   met_t * met0,
00614   met_t * met1,
00615   atm_t * atm,
00616   int ip) {
00617
00618   static double *iso, *ps, t, *ts;
00619
00620   static int idx, ip2, n;
00621
00622   FILE *in;
00623
00624   char line[LEN];
00625
00626   /* Initialize... */
00627   if (ip < 0) {
00628
00629     /* Allocate... */
00630     ALLOC(iso, double,
00631           NP);
00632     ALLOC(ps, double,
00633           NP);
00634     ALLOC(ts, double,
00635           NP);
00636
00637     /* Save pressure... */
00638     if (ctl->isosurf == 1)
00639       for (ip2 = 0; ip2 < atm->np; ip2++)
00640         iso[ip2] = atm->p[ip2];
00641
00642     /* Save density... */
00643     else if (ctl->isosurf == 2)
00644       for (ip2 = 0; ip2 < atm->np; ip2++) {
00645         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00646                         atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00647                         &t, NULL, NULL, NULL, NULL, NULL);
00648         iso[ip2] = atm->p[ip2] / t;
00649       }
00650
00651     /* Save potential temperature... */
00652     else if (ctl->isosurf == 3)
00653       for (ip2 = 0; ip2 < atm->np; ip2++) {
00654         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00655                         atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00656                         &t, NULL, NULL, NULL, NULL, NULL);
00657         iso[ip2] = THETA(atm->p[ip2], t);
00658       }
00659
00660     /* Read balloon pressure data... */
00661     else if (ctl->isosurf == 4) {
00662
00663       /* Write info... */
00664       printf("Read balloon pressure data: %s\n", ctl->balloon);
00665
00666       /* Open file... */
00667       if (!(in = fopen(ctl->balloon, "r")))
00668         ERRMSG("Cannot open file!");
00669
00670       /* Read pressure time series... */
00671       while (fgets(line, LEN, in))
00672         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00673           if ((++n) > NP)
00674             ERRMSG("Too many data points!");
00675
00676       /* Check number of points... */
00677       if (n < 1)
00678         ERRMSG("Could not read any data!");
00679
00680       /* Close file... */
00681       fclose(in);
00682     }
```

```
00683
00684     /* Leave initialization... */
00685     return;
00686   }
00687
00688   /* Restore pressure... */
00689   if (ctl->isosurf == 1)
00690     atm->p[ip] = iso[ip];
00691
00692   /* Restore density... */
00693   else if (ctl->isosurf == 2) {
00694     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00695                     atm->lat[ip], NULL, NULL, NULL, &t,
00696                     NULL, NULL, NULL, NULL, NULL);
00697     atm->p[ip] = iso[ip] * t;
00698   }
00699
00700   /* Restore potential temperature... */
00701   else if (ctl->isosurf == 3) {
00702     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00703                     atm->lat[ip], NULL, NULL, NULL, &t,
00704                     NULL, NULL, NULL, NULL, NULL);
00705     atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00706   }
00707
00708   /* Interpolate pressure... */
00709   else if (ctl->isosurf == 4) {
00710     if (atm->time[ip] <= ts[0])
00711       atm->p[ip] = ps[0];
00712     else if (atm->time[ip] >= ts[n - 1])
00713       atm->p[ip] = ps[n - 1];
00714     else {
00715       idx = locate(ts, n, atm->time[ip]);
00716       atm->p[ip] = LIN(ts[idx], ps[idx],
00717                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00718     }
00719   }
00720 }
00721
00722 /*****************************************************************************/
00723
00724 void module_meteo(
00725   ctl_t * ctl,
00726   met_t * met0,
00727   met_t * met1,
00728   atm_t * atm,
00729   int ip) {
00730
00731   static FILE *in;
00732
00733   static char filename[2 * LEN], line[LEN];
00734
00735   static double lon[GX], lat[GY], var[GX][GY],
00736     rdum, rlat, rlat_old = -999, rlon, rvar;
00737
00738   static int year_old, mon_old, day_old, nlon, nlat;
00739
00740   double a, b, c, dp, dx, dy, dtdp, dtdx, dtdy, dudp, dudy, dvdp, dvdx,
00741     lat0, lat1, latr, lon0, lon1, ps, pt, p0, p1, p_hno3, p_h2o,
00742     t, t0, t1, u, u0, u1, v, v0, v1, w, x1, x2, h2o, o3, vort, var0, var1, z;
00743
00744   int day, mon, year, idum, ilat, ilon;
00745
00746   /* Interpolate meteorological data... */
00747   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00748                   atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &h2o, &o3);
00749
00750   /* Set surface pressure... */
00751   if (ctl->qnt_ps >= 0)
00752     atm->q[ctl->qnt_ps][ip] = ps;
00753
00754   /* Set tropopause pressure... */
00755   if (ctl->qnt_pt >= 0)
00756     atm->q[ctl->qnt_pt][ip] = pt;
00757
00758   /* Set pressure... */
00759   if (ctl->qnt_p >= 0)
00760     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00761
00762   /* Set geopotential height... */
00763   if (ctl->qnt_z >= 0)
00764     atm->q[ctl->qnt_z][ip] = z;
00765
00766   /* Set temperature... */
```

```
00767   if (ctl->qnt_t >= 0)
00768     atm->q[ctl->qnt_t][ip] = t;
00769
00770   /* Set zonal wind... */
00771   if (ctl->qnt_u >= 0)
00772     atm->q[ctl->qnt_u][ip] = u;
00773
00774   /* Set meridional wind... */
00775   if (ctl->qnt_v >= 0)
00776     atm->q[ctl->qnt_v][ip] = v;
00777
00778   /* Set vertical velocity... */
00779   if (ctl->qnt_w >= 0)
00780     atm->q[ctl->qnt_w][ip] = w;
00781
00782   /* Set water vapor vmr... */
00783   if (ctl->qnt_h2o >= 0)
00784     atm->q[ctl->qnt_h2o][ip] = h2o;
00785
00786   /* Set ozone vmr... */
00787   if (ctl->qnt_o3 >= 0)
00788     atm->q[ctl->qnt_o3][ip] = o3;
00789
00790   /* Calculate potential temperature... */
00791   if (ctl->qnt_theta >= 0)
00792     atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00793
00794   /* Calculate potential vorticity... */
00795   if (ctl->qnt_pv >= 0) {
00796
00797     /* Get gradients in longitude... */
00798     latr = GSL_MIN(GSL_MAX(atm->lat[ip], -89.), 89.);
00799     lon0 = atm->lon[ip] - (met0->lon[1] - met0->lon[0]);
00800     lon1 = atm->lon[ip] + (met0->lon[1] - met0->lon[0]);
00801     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], lon0, latr,
00802                     NULL, NULL, NULL, &t0, NULL, &v0, NULL, NULL, NULL);
00803     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], lon1, latr,
00804                     NULL, NULL, NULL, &t1, NULL, &v1, NULL, NULL, NULL);
00805     dx = 1000. * deg2dx(lon1 - lon0, latr);
00806     dtdx = (THETA(atm->p[ip], t1) - THETA(atm->p[ip], t0)) / dx;
00807     dvdx = (v1 - v0) / dx;
00808
00809     /* Get gradients in latitude... */
00810     lat0 = latr - (met0->lat[1] - met0->lat[0]);
00811     lat1 = latr + (met0->lat[1] - met0->lat[0]);
00812     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip], lat0,
00813                     NULL, NULL, NULL, &t0, &u0, NULL, NULL, NULL, NULL);
00814     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip], lat1,
00815                     NULL, NULL, NULL, &t1, &u1, NULL, NULL, NULL, NULL);
00816     dy = 1000. * deg2dy(lat1 - lat0);
00817     dtdy = (THETA(atm->p[ip], t1) - THETA(atm->p[ip], t0)) / dy;
00818     dudy = (u1 * cos(lat1 / 180. * M_PI) - u0 * cos(lat0 / 180. * M_PI)) / dy;
00819
00820     /* Get gradients in pressure... */
00821     p0 = atm->p[ip] * 0.93;
00822     p1 = atm->p[ip] / 0.93;
00823     intpol_met_time(met0, met1, atm->time[ip], p0, atm->lon[ip], latr,
00824                     NULL, NULL, NULL, &t0, &u0, &v0, NULL, NULL, NULL);
00825     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip], latr,
00826                     NULL, NULL, NULL, &t1, &u1, &v1, NULL, NULL, NULL);
00827     dp = 100. * (p1 - p0);
00828     dtdp = (THETA(p1, t1) - THETA(p0, t0)) / dp;
00829     dudp = (u1 - u0) / dp;
00830     dvdp = (v1 - v0) / dp;
00831
00832     /* Set vorticity... */
00833     vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
00834
00835     /* Calculate PV... */
00836     atm->q[ctl->qnt_pv][ip] = 1e6 * G0 *
00837       (-dtdp * (dvdx - dudy / cos(latr / 180. * M_PI) + vort) +
00838        dvdp * dtdx - dudp * dtdy);
00839   }
00840
00841   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00842   if (ctl->qnt_tice >= 0)
00843     atm->q[ctl->qnt_tice][ip] =
00844       -2663.5 /
00845       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00846        12.537);
00847
00848   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00849   if (ctl->qnt_tnat >= 0) {
00850     if (ctl->psc_hno3 > 0)
00851       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
```

```
00852     else
00853       p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
00854         * 1e-9 * atm->p[ip] / 1.333224;
00855     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00856     a = 0.009179 - 0.00088 * log10(p_h2o);
00857     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00858     c = -11397.0 / a;
00859     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00860     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00861     if (x1 > 0)
00862       atm->q[ctl->qnt_tnat][ip] = x1;
00863     if (x2 > 0)
00864       atm->q[ctl->qnt_tnat][ip] = x2;
00865   }
00866
00867   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00868   if (ctl->qnt_tsts >= 0) {
00869     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00870       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00871     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00872                                        + atm->q[ctl->qnt_tnat][ip]);
00873   }
00874
00875   /* Read variance data for current day... */
00876   if (ip == 0 && ctl->qnt_gw_var >= 0) {
00877     jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00878     if (year != year_old || mon != mon_old || day != day_old) {
00879       year_old = year;
00880       mon_old = mon;
00881       day_old = day;
00882       nlon = nlat = -1;
00883       sprintf(filename, "%s_%d_%02d_%02d.tab",
00884               ctl->gw_basename, year, mon, day);
00885       if ((in = fopen(filename, "r"))) {
00886         printf("Read gravity wave data: %s\n", filename);
00887         while (fgets(line, LEN, in)) {
00888           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00889             continue;
00890           if (rlat != rlat_old) {
00891             rlat_old = rlat;
00892             if ((++nlat) > GY)
00893               ERRMSG("Too many latitudes!");
00894             nlon = -1;
00895           }
00896           if ((++nlon) > GX)
00897             ERRMSG("Too many longitudes!");
00898           lon[nlon] = rlon;
00899           lat[nlat] = rlat;
00900           var[nlon][nlat] = GSL_MAX(0, rvar);
00901         }
00902         fclose(in);
00903         nlat++;
00904         nlon++;
00905       } else
00906         printf("Warning: Missing gravity wave data: %s\n", filename);
00907     }
00908   }
00909
00910   /* Interpolate variance data... */
00911   if (ctl->qnt_gw_var >= 0) {
00912     if (nlat >= 2 && nlon >= 2) {
00913       ilat = locate(lat, nlat, atm->lat[ip]);
00914       ilon = locate(lon, nlon, atm->lon[ip]);
00915       var0 = LIN(lat[ilat], var[ilon][ilat],
00916                  lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00917       var1 = LIN(lat[ilat], var[ilon + 1][ilat],
00918                  lat[ilat + 1], var[ilon + 1][ilat + 1], atm->lat[ip]);
00919       atm->q[ctl->qnt_gw_var][ip]
00920         = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00921     } else
00922       atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00923   }
00924 }
00925
00926 /*****************************************************************************/
00927
00928 void module_position(
00929   met_t * met0,
00930   met_t * met1,
00931   atm_t * atm,
00932   int ip) {
00933
00934   double ps;
00935
00936   /* Calculate modulo... */
00937   atm->lon[ip] = fmod(atm->lon[ip], 360);
00938   atm->lat[ip] = fmod(atm->lat[ip], 360);
```

```
00939
00940   /* Check latitude... */
00941   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00942     if (atm->lat[ip] > 90) {
00943       atm->lat[ip] = 180 - atm->lat[ip];
00944       atm->lon[ip] += 180;
00945     }
00946     if (atm->lat[ip] < -90) {
00947       atm->lat[ip] = -180 - atm->lat[ip];
00948       atm->lon[ip] += 180;
00949     }
00950   }
00951
00952   /* Check longitude... */
00953   while (atm->lon[ip] < -180)
00954     atm->lon[ip] += 360;
00955   while (atm->lon[ip] >= 180)
00956     atm->lon[ip] -= 360;
00957
00958   /* Get surface pressure... */
00959   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00960                   atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00961                   NULL, NULL, NULL, NULL, NULL);
00962
00963   /* Check pressure... */
00964   if (atm->p[ip] > ps)
00965     atm->p[ip] = ps;
00966   else if (atm->p[ip] < met0->p[met0->np - 1])
00967     atm->p[ip] = met0->p[met0->np - 1];
00968 }
00969
00970 /*****************************************************************************/
00971
00972 void module_sedi(
00973   ctl_t * ctl,
00974   met_t * met0,
00975   met_t * met1,
00976   atm_t * atm,
00977   int ip,
00978   double dt) {
00979
00980   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00981   const double A = 1.249, B = 0.42, C = 0.87;
00982
00983   /* Average mass of an air molecule [kg/molec]: */
00984   const double m = 4.8096e-26;
00985
00986   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00987
00988   /* Convert units... */
00989   p = 100 * atm->p[ip];
00990   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00991   rho_p = atm->q[ctl->qnt_rho][ip];
00992
00993   /* Get temperature... */
00994   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00995                   atm->lat[ip], NULL, NULL, NULL, &T,
00996                   NULL, NULL, NULL, NULL, NULL);
00997
00998   /* Density of dry air... */
00999   rho = p / (R0 * T);
01000
01001   /* Dynamic viscosity of air... */
01002   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01003
01004   /* Thermal velocity of an air molecule... */
01005   v = sqrt(8 * KB * T / (M_PI * m));
01006
01007   /* Mean free path of an air molecule... */
01008   lambda = 2 * eta / (rho * v);
01009
01010   /* Knudsen number for air... */
01011   K = lambda / r_p;
01012
01013   /* Cunningham slip-flow correction... */
01014   G = 1 + K * (A + B * exp(-C / K));
01015
01016   /* Sedimentation (fall) velocity... */
01017   v_p = 2. * gsl_pow_2(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01018
01019   /* Calculate pressure change... */
01020   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01021 }
01022
01023 /*****************************************************************************/
01024
```

```
01025 void write_output(
01026   const char *dirname,
01027   ctl_t * ctl,
01028   met_t * met0,
01029   met_t * met1,
01030   atm_t * atm,
01031   double t) {
01032
01033   char filename[2 * LEN];
01034
01035   double r;
01036
01037   int year, mon, day, hour, min, sec;
01038
01039   /* Get time... */
01040   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01041
01042   /* Write atmospheric data... */
01043   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01044     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01045             dirname, ctl->atm_basename, year, mon, day, hour, min);
01046     write_atm(filename, ctl, atm, t);
01047   }
01048
01049   /* Write CSI data... */
01050   if (ctl->csi_basename[0] != '-') {
01051     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01052     write_csi(filename, ctl, atm, t);
01053   }
01054
01055   /* Write ensemble data... */
01056   if (ctl->ens_basename[0] != '-') {
01057     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01058     write_ens(filename, ctl, atm, t);
01059   }
01060
01061   /* Write gridded data... */
01062   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01063     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01064             dirname, ctl->grid_basename, year, mon, day, hour, min);
01065     write_grid(filename, ctl, met0, met1, atm, t);
01066   }
01067
01068   /* Write profile data... */
01069   if (ctl->prof_basename[0] != '-') {
01070     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01071     write_prof(filename, ctl, met0, met1, atm, t);
01072   }
01073
01074   /* Write station data... */
01075   if (ctl->stat_basename[0] != '-') {
01076     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01077     write_station(filename, ctl, atm, t);
01078   }
01079 }
```

5.37 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[])

5.37.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.37.2 Function Documentation

5.37.2.1 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 }
```

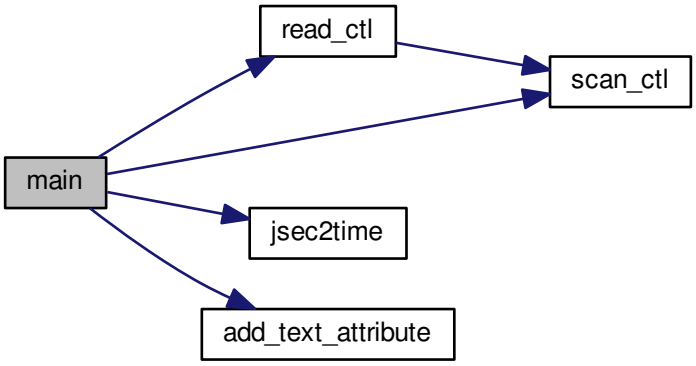
5.37.2.2 int main (int *argc*, char * *argv[ ]*)

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:



5.38 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