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:

https://github.com/slcs-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]
  Quantity 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].
- double cache_time [EX][EY][EP]

  Cache for reference time of wind standard deviations.
- float cache_usig [EX][EY][EP]

  Cache for zonal wind standard deviations.
- float cache_vsig [EX][EY][EP]

  Cache for meridional wind standard deviations.
- float cache_wsig [EX][EY][EP]

  Cache for vertical velocity standard deviations.

4.1.1 Detailed Description

Atmospheric data.

Definition at line 592 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

Definition at line 595 of file libtrac.h.

4.1.2.2 double atm_t::time[NP]

Time [s].

Definition at line 598 of file libtrac.h.

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

Definition at line 601 of file libtrac.h.

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 604 of file libtrac.h.

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 607 of file libtrac.h.

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

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

Definition at line 610 of file libtrac.h.

4.1.2.7 float atm_t::up[NP]

Zonal wind perturbation [m/s].

Definition at line 613 of file libtrac.h.

4.1.2.8 float atm_t::vp[NP]

Meridional wind perturbation [m/s].

Definition at line 616 of file libtrac.h.

4.1.2.9 float atm_t::wp[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 619 of file libtrac.h.

4.1.2.10 double atm_t::cache_time[EX][EY][EP]

Cache for reference time of wind standard deviations.

Definition at line 622 of file libtrac.h.

4.1.2.11 float atm_t::cache_usig[EX][EY][EP]

Cache for zonal wind standard deviations.

Definition at line 625 of file libtrac.h.

4.1.2.12 float atm_t::cache_vsig[EX][EY][EP]

Cache for meridional wind standard deviations.

Definition at line 628 of file libtrac.h.

4.1.2.13 float atm_t::cache_wsig[EX][EY][EP]

Cache for vertical velocity standard deviations.

Definition at line 631 of file libtrac.h.

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

- libtrac.h

4.2 ctl_t Struct Reference

Control parameters.

`#include <libtrac.h>`

Data Fields

- int nq
  Number of quantities.
- char qnt_name [NQ][LEN]
  Quantity names.
- char qnt_unit [NQ][LEN]
  Quantity units.
- char qnt_format [NQ][LEN]
  Quantity output format.
- int qnt_ens
  Quantity array index for ensemble IDs.
- int qnt_m
  Quantity array index for mass.
- int qnt_rho
  Quantity array index for particle density.
- int qnt_r
  Quantity array index for particle radius.
- int qnt_ps
  Quantity array index for surface pressure.
- int qnt_pt
  Quantity array index for tropopause pressure.
- int qnt_z
  Quantity array index for geopotential height.
- int qnt_p
  Quantity array index for pressure.
- int qnt_t
  Quantity array index for temperature.
- int qnt_u
  Quantity array index for zonal wind.
- int qnt_v
  Quantity array index for meridional wind.
- int qnt_w
  Quantity array index for vertical velocity.
- int qnt_h2o
  Quantity array index for water vapor vmr.
- int qnt_o3
  Quantity array index for ozone vmr.
- int qnt_theta
  Quantity array index for potential temperature.
- int qnt_vh
  Quantity array index for horizontal wind.
- int qnt_vz
  Quantity array index for vertical velocity.

- int qnt_pv

  Quantity array index for potential vorticity.
- int qnt_tice

  Quantity array index for T_ice.
- int qnt_tsts

  Quantity array index for T_STS.
- int qnt_tnat

  Quantity array index for T_NAT.
- int qnt_stat

  Quantity array index for station flag.
- int direction

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

  Start time of simulation [s].
- double t_stop

  Stop time of simulation [s].
- double dt_mod

  Time step of simulation [s].
- double dt_met

  Time step of meteorological data [s].
- int met_dx

  Stride for longitudes.
- int met_dy

  Stride for latitudes.
- int met_dp

  Stride for pressure levels.
- int met_sx

  Smoothing for longitudes.
- int met_sy

  Smoothing for latitudes.
- int met_sp

  Smoothing 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.
- double met_dt_out

  Time step for sampling of meteo data along trajectories [s].
- 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_mesox

  Horizontal scaling factor for mesoscale wind fluctuations.
- double turb_mesoz

  Vertical scaling factor for mesoscale wind fluctuations.
- double molmass

  Molar mass [g/mol].
- double tdec_trop

  Life time of particles (troposphere) [s].
- double tdec_strat

  Life time of particles (stratosphere) [s].
- double psc_h2o

  H2O volume mixing ratio for PSC analysis.
- double psc_hno3

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

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

  Gnuplot file for atmospheric data.
- double atm_dt_out

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

  Time filter for atmospheric data output (0=no, 1=yes).
- 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 273 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 276 of file libtrac.h.

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

Quantity names.

Definition at line 279 of file libtrac.h.

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

Quantity units.

Definition at line 282 of file libtrac.h.

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

Quantity output format.

Definition at line 285 of file libtrac.h.

4.2.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 288 of file libtrac.h.

4.2.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 291 of file libtrac.h.

4.2.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 294 of file libtrac.h.

4.2.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 297 of file libtrac.h.

4.2.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 300 of file libtrac.h.

4.2.2.10 int ctl_t::qnt_pt

Quantity array index for tropopause pressure.

Definition at line 303 of file libtrac.h.

4.2.2.11 int ctl_t::qnt_z

Quantity array index for geopotential height.

Definition at line 306 of file libtrac.h.

4.2.2.12 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 309 of file libtrac.h.

4.2.2.13 int ctl_t::qnt_t

Quantity array index for temperature.

Definition at line 312 of file libtrac.h.

4.2.2.14 int ctl_t::qnt_u

Quantity array index for zonal wind.

Definition at line 315 of file libtrac.h.

4.2.2.15 int ctl_t::qnt_v

Quantity array index for meridional wind.

Definition at line 318 of file libtrac.h.

4.2.2.16 int ctl_t::qnt_w

Quantity array index for vertical velocity.

Definition at line 321 of file libtrac.h.

4.2.2.17 int ctl_t::qnt_h2o

Quantity array index for water vapor vmr.

Definition at line 324 of file libtrac.h.

4.2.2.18 int ctl_t::qnt_o3

Quantity array index for ozone vmr.

Definition at line 327 of file libtrac.h.

4.2.2.19 int ctl_t::qnt_theta

Quantity array index for potential temperature.

Definition at line 330 of file libtrac.h.

4.2.2.20 int ctl_t::qnt_vh

Quantity array index for horizontal wind.

Definition at line 333 of file libtrac.h.

4.2.2.21 int ctl_t::qnt_vz

Quantity array index for vertical velocity.

Definition at line 336 of file libtrac.h.

4.2.2.22 int ctl_t::qnt_pv

Quantity array index for potential vorticity.

Definition at line 339 of file libtrac.h.

4.2.2.23 int ctl_t::qnt_tice

Quantity array index for T_ice.

Definition at line 342 of file libtrac.h.

4.2.2.24 int ctl_t::qnt_tsts

Quantity array index for T_STS.

Definition at line 345 of file libtrac.h.

4.2.2.25 int ctl_t::qnt_tnat

Quantity array index for T_NAT.

Definition at line 348 of file libtrac.h.

4.2.2.26 int ctl_t::qnt_stat

Quantity array index for station flag.

Definition at line 351 of file libtrac.h.

4.2.2.27 int ctl_t::direction

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

Definition at line 354 of file libtrac.h.

4.2.2.28 double ctl_t::t_start

Start time of simulation [s].

Definition at line 357 of file libtrac.h.

4.2.2.29 double ctl_t::t_stop

Stop time of simulation [s].

Definition at line 360 of file libtrac.h.

4.2.2.30 double ctl_t::dt_mod

Time step of simulation [s].

Definition at line 363 of file libtrac.h.

4.2.2.31 double ctl_t::dt_met

Time step of meteorological data [s].

Definition at line 366 of file libtrac.h.

4.2.2.32 int ctl_t::met_dx

Stride for longitudes.

Definition at line 369 of file libtrac.h.

4.2.2.33 int ctl_t::met_dy

Stride for latitudes.

Definition at line 372 of file libtrac.h.

4.2.2.34 int ctl_t::met_dp

Stride for pressure levels.

Definition at line 375 of file libtrac.h.

4.2.2.35 int ctl_t::met_sx

Smoothing for longitudes.

Definition at line 378 of file libtrac.h.

4.2.2.36 int ctl_t::met_sy

Smoothing for latitudes.

Definition at line 381 of file libtrac.h.

4.2.2.37 int ctl_t::met_sp

Smoothing for pressure levels.

Definition at line 384 of file libtrac.h.

4.2.2.38 int ctl_t::met_np

Number of target pressure levels.

Definition at line 387 of file libtrac.h.

4.2.2.39 double ctl_t::met_p[EP]

Target pressure levels [hPa].

Definition at line 390 of file libtrac.h.

4.2.2.40 int ctl_t::met_tropo

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

Definition at line 394 of file libtrac.h.

4.2.2.41 char ctl_t::met_geopot[LEN]

Surface geopotential data file.

Definition at line 397 of file libtrac.h.

4.2.2.42 double ctl_t::met_dt_out

Time step for sampling of meteo data along trajectories [s].

Definition at line 400 of file libtrac.h.

4.2.2.43 char ctl_t::met_stage[LEN]

Command to stage meteo data.

Definition at line 403 of file libtrac.h.

4.2.2.44 int ctl_t::isosurf

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

Definition at line 407 of file libtrac.h.

4.2.2.45 char ctl_t::balloon[LEN]

Balloon position filename.

Definition at line 410 of file libtrac.h.

4.2.2.46 double ctl_t::turb_dx_trop

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

Definition at line 413 of file libtrac.h.

4.2.2.47 double ctl_t::turb_dx_strat

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

Definition at line 416 of file libtrac.h.

4.2.2.48 double ctl_t::turb_dz_trop

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

Definition at line 419 of file libtrac.h.

4.2.2.49 double ctl_t::turb_dz_strat

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

Definition at line 422 of file libtrac.h.

4.2.2.50 double ctl_t::turb_mesox

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 425 of file libtrac.h.

4.2.2.51 double ctl_t::turb_mesoz

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 428 of file libtrac.h.

4.2.2.52 double ctl_t::molmass

Molar mass [g/mol].

Definition at line 431 of file libtrac.h.

4.2.2.53 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 434 of file libtrac.h.

4.2.2.54 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 437 of file libtrac.h.

4.2.2.55 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 440 of file libtrac.h.

4.2.2.56 double ctl_t::psc_hno3

HNO3 volume mixing ratio for PSC analysis.

Definition at line 443 of file libtrac.h.

4.2.2.57 char ctl_t::atm_basename[LEN]

Basename of atmospheric data files.

Definition at line 446 of file libtrac.h.

4.2.2.58 char ctl_t::atm_gpfile[LEN]

Gnuplot file for atmospheric data.

Definition at line 449 of file libtrac.h.

4.2.2.59 double ctl_t::atm_dt_out

Time step for atmospheric data output [s].

Definition at line 452 of file libtrac.h.

4.2.2.60 int ctl_t::atm_filter

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

Definition at line 455 of file libtrac.h.

4.2.2.61 int ctl_t::atm_type

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

Definition at line 458 of file libtrac.h.

4.2.2.62 char ctl_t::csi_basename[LEN]

Basename of CSI data files.

Definition at line 461 of file libtrac.h.

4.2.2.63 double ctl_t::csi_dt_out

Time step for CSI data output [s].

Definition at line 464 of file libtrac.h.

4.2.2.64 char ctl_t::csi_obsfile[LEN]

Observation data file for CSI analysis.

Definition at line 467 of file libtrac.h.

4.2.2.65 double ctl_t::csi_obsmin

Minimum observation index to trigger detection.

Definition at line 470 of file libtrac.h.

4.2.2.66 double ctl_t::csi_modmin

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

Definition at line 473 of file libtrac.h.

4.2.2.67 int ctl_t::csi_nz

Number of altitudes of gridded CSI data.

Definition at line 476 of file libtrac.h.

4.2.2.68 double ctl_t::csi_z0

Lower altitude of gridded CSI data [km].

Definition at line 479 of file libtrac.h.

4.2.2.69 double ctl_t::csi_z1

Upper altitude of gridded CSI data [km].

Definition at line 482 of file libtrac.h.

4.2.2.70 int ctl_t::csi_nx

Number of longitudes of gridded CSI data.

Definition at line 485 of file libtrac.h.

4.2.2.71 double ctl_t::csi_lon0

Lower longitude of gridded CSI data [deg].

Definition at line 488 of file libtrac.h.

4.2.2.72 double ctl_t::csi_lon1

Upper longitude of gridded CSI data [deg].

Definition at line 491 of file libtrac.h.

4.2.2.73 int ctl_t::csi_ny

Number of latitudes of gridded CSI data.

Definition at line 494 of file libtrac.h.

4.2.2.74 double ctl_t::csi_lat0

Lower latitude of gridded CSI data [deg].

Definition at line 497 of file libtrac.h.

4.2.2.75 double ctl_t::csi_lat1

Upper latitude of gridded CSI data [deg].

Definition at line 500 of file libtrac.h.

4.2.2.76 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 503 of file libtrac.h.

4.2.2.77 char ctl_t::grid_gpfile[LEN]

Gnuplot file for gridded data.

Definition at line 506 of file libtrac.h.

4.2.2.78 double ctl_t::grid_dt_out

Time step for gridded data output [s].

Definition at line 509 of file libtrac.h.

4.2.2.79 int ctl_t::grid_sparse

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

Definition at line 512 of file libtrac.h.

4.2.2.80 int ctl_t::grid_nz

Number of altitudes of gridded data.

Definition at line 515 of file libtrac.h.

4.2.2.81 double ctl_t::grid_z0

Lower altitude of gridded data [km].

Definition at line 518 of file libtrac.h.

4.2.2.82 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 521 of file libtrac.h.

4.2.2.83 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 524 of file libtrac.h.

4.2.2.84 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 527 of file libtrac.h.

4.2.2.85 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 530 of file libtrac.h.

4.2.2.86 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 533 of file libtrac.h.

4.2.2.87 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 536 of file libtrac.h.

4.2.2.88 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 539 of file libtrac.h.

4.2.2.89 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 542 of file libtrac.h.

4.2.2.90 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 545 of file libtrac.h.

4.2.2.91 int ctl_t::prof_nz

Number of altitudes of gridded profile data.

Definition at line 548 of file libtrac.h.

4.2.2.92 double ctl_t::prof_z0

Lower altitude of gridded profile data [km].

Definition at line 551 of file libtrac.h.

4.2.2.93 double ctl_t::prof_z1

Upper altitude of gridded profile data [km].

Definition at line 554 of file libtrac.h.

4.2.2.94 int ctl_t::prof_nx

Number of longitudes of gridded profile data.

Definition at line 557 of file libtrac.h.

4.2.2.95 double ctl_t::prof_lon0

Lower longitude of gridded profile data [deg].

Definition at line 560 of file libtrac.h.

4.2.2.96 double ctl_t::prof_lon1

Upper longitude of gridded profile data [deg].

Definition at line 563 of file libtrac.h.

4.2.2.97 int ctl_t::prof_ny

Number of latitudes of gridded profile data.

Definition at line 566 of file libtrac.h.

4.2.2.98 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 569 of file libtrac.h.

4.2.2.99 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 572 of file libtrac.h.

4.2.2.100 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 575 of file libtrac.h.

4.2.2.101 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 578 of file libtrac.h.

4.2.2.102 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 581 of file libtrac.h.

4.2.2.103 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 584 of file libtrac.h.

4.2.2.104 double ctl_t::stat_r

Search radius around station [km].

Definition at line 587 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 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 pv [EX][EY][EP]
  Potential vorticity [PVU].
- float h2o [EX][EY][EP]
  Water vapor volume mixing ratio [1].
- float o3 [EX][EY][EP]
  Ozone volume mixing ratio [1].
- float pl [EX][EY][EP]
  Pressure on model levels [hPa].

4.3.1 Detailed Description

Meteorological data.

Definition at line 636 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 double met_t::time

Time [s].

Definition at line 639 of file libtrac.h.

4.3.2.2 int met_t::nx

Number of longitudes.

Definition at line 642 of file libtrac.h.

4.3.2.3 int met_t::ny

Number of latitudes.

Definition at line 645 of file libtrac.h.

4.3.2.4 int met_t::np

Number of pressure levels.

Definition at line 648 of file libtrac.h.

4.3.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 651 of file libtrac.h.

4.3.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 654 of file libtrac.h.

4.3.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 657 of file libtrac.h.

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

Surface pressure [hPa].

Definition at line 660 of file libtrac.h.

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

Tropopause pressure [hPa].

Definition at line 663 of file libtrac.h.

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

Geopotential height [km].

Definition at line 666 of file libtrac.h.

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

Temperature [K].

Definition at line 669 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 672 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 675 of file libtrac.h.

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

Vertical wind [hPa/s].

Definition at line 678 of file libtrac.h.

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

Potential vorticity [PVU].

Definition at line 681 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 684 of file libtrac.h.

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

Ozone volume mixing ratio [1].

Definition at line 687 of file libtrac.h.

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

Pressure on model levels [hPa].

Definition at line 690 of file libtrac.h.

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

- libtrac.h

5 File Documentation

5.1 atm_conv.c File Reference

Convert file format of air parcel data files.

Functions

- int main (int argc, char ∗argv[ ])

5.1.1 Detailed Description

Convert file format of air parcel data files.

Definition in file atm_conv.c.

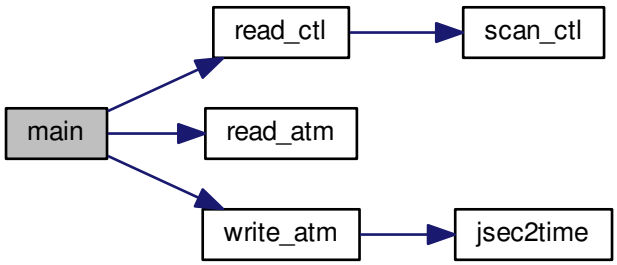
5.1.2 Function Documentation

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

Definition at line 27 of file atm_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   if (!read_atm(argv[2], &ctl, atm))
00049     ERRMSG("Cannot open file!");
00050
00051   /* Write atmospheric data... */
00052   ctl.atm_type = atoi(argv[5]);
00053   write_atm(argv[4], &ctl, atm, 0);
00054
00055   /* Free... */
00056   free(atm);
00057
00058   return EXIT_SUCCESS;
00059 }
```

Here is the call graph for this function:



5.2 atm_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   Copyright (C) 2013-2019 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   if (!read_atm(argv[2], &ctl, atm))
00049     ERRMSG("Cannot open file!");
00050
00051   /* Write atmospheric data... */
00052   ctl.atm_type = atoi(argv[5]);
00053   write_atm(argv[4], &ctl, atm, 0);
00054
00055   /* Free... */
00056   free(atm);
00057
00058   return EXIT_SUCCESS;
00059 }
```

5.3 atm_dist.c File Reference

Calculate transport deviations of trajectories.

Functions

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

5.3.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file atm_dist.c.

5.3.2 Function Documentation

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

Definition at line 27 of file atm_dist.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
```

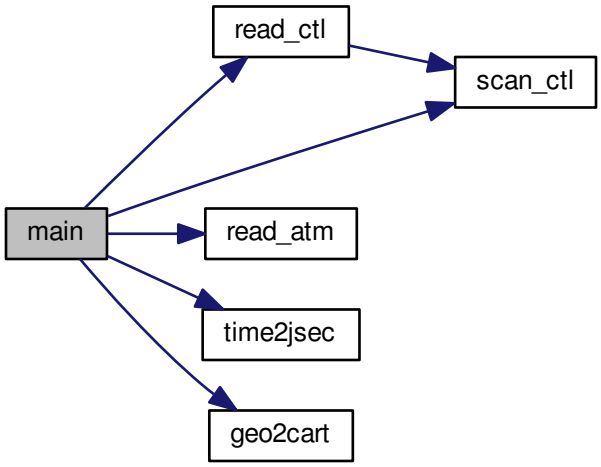
```
00039   double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
00040     *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041     *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00042     t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;
00043
00044   int ens, f, init = 0, 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(ahtd, double,
00070         NP);
00071   ALLOC(avtd, double,
00072         NP);
00073   ALLOC(aqtd, double,
00074         NP * NQ);
00075   ALLOC(rhtd, double,
00076         NP);
00077   ALLOC(rvtd, double,
00078         NP);
00079   ALLOC(rqtd, double,
00080         NP * NQ);
00081   ALLOC(work, double,
00082         NP);
00083
00084   /* Check arguments... */
00085   if (argc < 6)
00086     ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atm1a> <atm1b>"
00087            " [<atm2a> <atm2b> ...]");
00088
00089   /* Read control parameters... */
00090   read_ctl(argv[1], argc, argv, &ctl);
00091   ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092   p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093   p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094   lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095   lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096   lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097   lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099   /* Write info... */
00100   printf("Write transport deviations: %s\n", argv[2]);
00101
00102   /* Create output file... */
00103   if (!(out = fopen(argv[2], "w")))
00104     ERRMSG("Cannot create file!");
00105
00106   /* Write header... */
00107   fprintf(out,
00108           "# $1 = time [s]\n"
00109           "# $2 = time difference [s]\n"
00110           "# $3 = absolute horizontal distance (%s) [km]\n"
00111           "# $4 = relative horizontal distance (%s) [%%]\n"
00112           "# $5 = absolute vertical distance (%s) [km]\n"
00113           "# $6 = relative vertical distance (%s) [%%]\n",
00114           argv[3], argv[3], argv[3], argv[3]);
00115   for (iq = 0; iq < ctl.nq; iq++)
00116     fprintf(out,
00117             "# $%d = %s absolute difference (%s) [%s]\n"
00118             "# $%d = %s relative difference (%s) [%%]\n",
00119             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121   fprintf(out, "# $%d = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123   /* Loop over file pairs... */
00124   for (f = 4; f < argc; f += 2) {
00125
```

```
00126     /* Read atmopheric data... */
00127     if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00128       continue;
00129
00130     /* Check if structs match... */
00131     if (atm1->np != atm2->np)
00132       ERRMSG("Different numbers of particles!");
00133
00134     /* Get time from filename... */
00135     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00136     year = atoi(tstr);
00137     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00138     mon = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00140     day = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00142     hour = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00144     min = atoi(tstr);
00145     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00146
00147     /* Save initial time... */
00148     if (!init) {
00149       init = 1;
00150       t0 = t;
00151     }
00152
00153     /* Init... */
00154     np = 0;
00155     for (ip = 0; ip < atm1->np; ip++) {
00156       ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00157       for (iq = 0; iq < ctl.nq; iq++)
00158         aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00159     }
00160
00161     /* Loop over air parcels... */
00162     for (ip = 0; ip < atm1->np; ip++) {
00163
00164       /* Check data... */
00165       if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00166         continue;
00167
00168       /* Check ensemble index... */
00169       if (ctl.qnt_ens > 0
00170           && (atm1->q[ctl.qnt_ens][ip] != ens
00171               || atm2->q[ctl.qnt_ens][ip] != ens))
00172         continue;
00173
00174       /* Check spatial range... */
00175       if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00176           || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00177           || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00178         continue;
00179       if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00180           || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00181           || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00182         continue;
00183
00184       /* Convert coordinates... */
00185       geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00186       geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00187       z1 = Z(atm1->p[ip]);
00188       z2 = Z(atm2->p[ip]);
00189
00190       /* Calculate absolute transport deviations... */
00191       ahtd[np] = DIST(x1, x2);
00192       avtd[np] = z1 - z2;
00193       for (iq = 0; iq < ctl.nq; iq++)
00194         aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00195
00196       /* Calculate relative transport deviations... */
00197       if (f > 4) {
00198
00199         /* Get trajectory lengths... */
00200         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00201         lh1[ip] += DIST(x0, x1);
00202         lv1[ip] += fabs(z1_old[ip] - z1);
00203
00204         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00205         lh2[ip] += DIST(x0, x2);
00206         lv2[ip] += fabs(z2_old[ip] - z2);
00207
00208         /* Get relative transport deviations... */
00209         if (lh1[ip] + lh2[ip] > 0)
00210           rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00211         if (lv1[ip] + lv2[ip] > 0)
00212           rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
```

```
00213         for (iq = 0; iq < ctl.nq; iq++)
00214           rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00215             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00216       }
00217
00218       /* Save positions of air parcels... */
00219       lon1_old[ip] = atm1->lon[ip];
00220       lat1_old[ip] = atm1->lat[ip];
00221       z1_old[ip] = z1;
00222
00223       lon2_old[ip] = atm2->lon[ip];
00224       lat2_old[ip] = atm2->lat[ip];
00225       z2_old[ip] = z2;
00226
00227       /* Increment air parcel counter... */
00228       np++;
00229     }
00230
00231     /* Get statistics... */
00232     if (strcasecmp(argv[3], "mean") == 0) {
00233       ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00234       rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00235       avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00236       rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00237       for (iq = 0; iq < ctl.nq; iq++) {
00238         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00239         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00240       }
00241     } else if (strcasecmp(argv[3], "stddev") == 0) {
00242       ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00243       rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00244       avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00245       rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00246       for (iq = 0; iq < ctl.nq; iq++) {
00247         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00248         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00249       }
00250     } else if (strcasecmp(argv[3], "min") == 0) {
00251       ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00252       rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00253       avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00254       rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00255       for (iq = 0; iq < ctl.nq; iq++) {
00256         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00257         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00258       }
00259     } else if (strcasecmp(argv[3], "max") == 0) {
00260       ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00261       rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00262       avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00263       rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00264       for (iq = 0; iq < ctl.nq; iq++) {
00265         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00266         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00267       }
00268     } else if (strcasecmp(argv[3], "skew") == 0) {
00269       ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00270       rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00271       avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00272       rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00273       for (iq = 0; iq < ctl.nq; iq++) {
00274         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00275         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00276       }
00277     } else if (strcasecmp(argv[3], "kurt") == 0) {
00278       ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00279       rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00280       avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00281       rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00282       for (iq = 0; iq < ctl.nq; iq++) {
00283         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00284         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00285       }
00286     } else if (strcasecmp(argv[3], "median") == 0) {
00287       ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00288       rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00289       avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00290       rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00291       for (iq = 0; iq < ctl.nq; iq++) {
00292         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00293         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00294       }
00295     } else if (strcasecmp(argv[3], "absdev") == 0) {
00296       ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
00297       rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
00298       avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
00299       rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
```

```
00300       for (iq = 0; iq < ctl.nq; iq++) {
00301         aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
00302         rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
00303       }
00304     } else if (strcasecmp(argv[3], "mad") == 0) {
00305       ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00306       rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00307       avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00308       rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00309       for (iq = 0; iq < ctl.nq; iq++) {
00310         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00311         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00312       }
00313     } else
00314       ERRMSG("Unknown parameter!");
00315
00316     /* Write output... */
00317     fprintf(out, "%.2f %.2f %g %g %g %g",
00318             ahtdm, rhtdm, avtdm, rvtdm);
00319     for (iq = 0; iq < ctl.nq; iq++) {
00320       fprintf(out, " ");
00321       fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00322       fprintf(out, " ");
00323       fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00324     }
00325     fprintf(out, " %d\n", np);
00326   }
00327
00328   /* Close file... */
00329   fclose(out);
00330
00331   /* Free... */
00332   free(atm1);
00333   free(atm2);
00334   free(lon1_old);
00335   free(lat1_old);
00336   free(z1_old);
00337   free(lh1);
00338   free(lv1);
00339   free(lon2_old);
00340   free(lat2_old);
00341   free(z2_old);
00342   free(lh2);
00343   free(lv2);
00344   free(ahtd);
00345   free(avtd);
00346   free(aqtd);
00347   free(rhtd);
00348   free(rvtd);
00349   free(rqtd);
00350   free(work);
00351
00352   return EXIT_SUCCESS;
00353 }
```

Here is the call graph for this function:



5.4 atm_dist.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 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, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
00040     *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041     *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00042     t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;
00043
00044   int ens, f, init = 0, 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(ahtd, double,
00070         NP);
00071   ALLOC(avtd, double,
00072         NP);
00073   ALLOC(aqtd, double,
00074         NP * NQ);
00075   ALLOC(rhtd, double,
00076         NP);
00077   ALLOC(rvtd, double,
00078         NP);
00079   ALLOC(rqtd, double,
00080         NP * NQ);
00081   ALLOC(work, double,
00082         NP);
00083
00084   /* Check arguments... */
00085   if (argc < 6)
00086     ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atm1a> <atm1b>"
00087            " [<atm2a> <atm2b> ...]");
00088
00089   /* Read control parameters... */
00090   read_ctl(argv[1], argc, argv, &ctl);
00091   ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092   p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093   p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094   lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095   lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096   lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097   lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099   /* Write info... */
00100   printf("Write transport deviations: %s\n", argv[2]);
00101
00102   /* Create output file... */
00103   if (!(out = fopen(argv[2], "w")))
00104     ERRMSG("Cannot create file!");
00105
00106   /* Write header... */
00107   fprintf(out,
00108           "# $1 = time [s]\n"
00109           "# $2 = time difference [s]\n"
00110           "# $3 = absolute horizontal distance (%s) [km]\n"
00111           "# $4 = relative horizontal distance (%s) [%%]\n"
00112           "# $5 = absolute vertical distance (%s) [km]\n"
00113           "# $6 = relative vertical distance (%s) [%%]\n",
00114           argv[3], argv[3], argv[3], argv[3]);
00115   for (iq = 0; iq < ctl.nq; iq++)
00116     fprintf(out,
00117             "# $%d = %s absolute difference (%s) [%s]\n"
00118             "# $%d = %s relative difference (%s) [%%]\n",
00119             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121   fprintf(out, "# $%d = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123   /* Loop over file pairs... */
00124   for (f = 4; f < argc; f += 2) {
00125
00126     /* Read atmopheric data... */
00127     if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00128       continue;
00129
00130     /* Check if structs match... */
00131     if (atm1->np != atm2->np)
00132       ERRMSG("Different numbers of particles!");
00133
00134     /* Get time from filename... */
00135     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00136     year = atoi(tstr);
```

```
00137     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00138     mon = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00140     day = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00142     hour = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00144     min = atoi(tstr);
00145     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00146
00147     /* Save initial time... */
00148     if (!init) {
00149       init = 1;
00150       t0 = t;
00151     }
00152
00153     /* Init... */
00154     np = 0;
00155     for (ip = 0; ip < atm1->np; ip++) {
00156       ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00157       for (iq = 0; iq < ctl.nq; iq++)
00158         aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00159     }
00160
00161     /* Loop over air parcels... */
00162     for (ip = 0; ip < atm1->np; ip++) {
00163
00164       /* Check data... */
00165       if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00166         continue;
00167
00168       /* Check ensemble index... */
00169       if (ctl.qnt_ens > 0
00170           && (atm1->q[ctl.qnt_ens][ip] != ens
00171               || atm2->q[ctl.qnt_ens][ip] != ens))
00172         continue;
00173
00174       /* Check spatial range... */
00175       if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00176           || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00177           || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00178         continue;
00179       if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00180           || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00181           || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00182         continue;
00183
00184       /* Convert coordinates... */
00185       geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00186       geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00187       z1 = Z(atm1->p[ip]);
00188       z2 = Z(atm2->p[ip]);
00189
00190       /* Calculate absolute transport deviations... */
00191       ahtd[np] = DIST(x1, x2);
00192       avtd[np] = z1 - z2;
00193       for (iq = 0; iq < ctl.nq; iq++)
00194         aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00195
00196       /* Calculate relative transport deviations... */
00197       if (f > 4) {
00198
00199         /* Get trajectory lengths... */
00200         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00201         lh1[ip] += DIST(x0, x1);
00202         lv1[ip] += fabs(z1_old[ip] - z1);
00203
00204         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00205         lh2[ip] += DIST(x0, x2);
00206         lv2[ip] += fabs(z2_old[ip] - z2);
00207
00208         /* Get relative transport deviations... */
00209         if (lh1[ip] + lh2[ip] > 0)
00210           rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00211         if (lv1[ip] + lv2[ip] > 0)
00212           rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00213         for (iq = 0; iq < ctl.nq; iq++)
00214           rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00215             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00216       }
00217
00218       /* Save positions of air parcels... */
00219       lon1_old[ip] = atm1->lon[ip];
00220       lat1_old[ip] = atm1->lat[ip];
00221       z1_old[ip] = z1;
00222
00223       lon2_old[ip] = atm2->lon[ip];
```

```
00224       lat2_old[ip] = atm2->lat[ip];
00225       z2_old[ip] = z2;
00226
00227       /* Increment air parcel counter... */
00228       np++;
00229     }
00230
00231     /* Get statistics... */
00232     if (strcasecmp(argv[3], "mean") == 0) {
00233       ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00234       rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00235       avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00236       rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00237       for (iq = 0; iq < ctl.nq; iq++) {
00238         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00239         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00240       }
00241     } else if (strcasecmp(argv[3], "stddev") == 0) {
00242       ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00243       rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00244       avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00245       rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00246       for (iq = 0; iq < ctl.nq; iq++) {
00247         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00248         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00249       }
00250     } else if (strcasecmp(argv[3], "min") == 0) {
00251       ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00252       rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00253       avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00254       rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00255       for (iq = 0; iq < ctl.nq; iq++) {
00256         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00257         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00258       }
00259     } else if (strcasecmp(argv[3], "max") == 0) {
00260       ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00261       rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00262       avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00263       rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00264       for (iq = 0; iq < ctl.nq; iq++) {
00265         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00266         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00267       }
00268     } else if (strcasecmp(argv[3], "skew") == 0) {
00269       ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00270       rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00271       avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00272       rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00273       for (iq = 0; iq < ctl.nq; iq++) {
00274         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00275         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00276       }
00277     } else if (strcasecmp(argv[3], "kurt") == 0) {
00278       ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00279       rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00280       avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00281       rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00282       for (iq = 0; iq < ctl.nq; iq++) {
00283         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00284         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00285       }
00286     } else if (strcasecmp(argv[3], "median") == 0) {
00287       ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00288       rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00289       avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00290       rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00291       for (iq = 0; iq < ctl.nq; iq++) {
00292         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00293         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00294       }
00295     } else if (strcasecmp(argv[3], "absdev") == 0) {
00296       ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
00297       rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
00298       avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
00299       rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
00300       for (iq = 0; iq < ctl.nq; iq++) {
00301         aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
00302         rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
00303       }
00304     } else if (strcasecmp(argv[3], "mad") == 0) {
00305       ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00306       rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00307       avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00308       rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00309       for (iq = 0; iq < ctl.nq; iq++) {
00310         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
```

```
00311         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00312       }
00313     } else
00314       ERRMSG("Unknown parameter!");
00315
00316     /* Write output... */
00317     fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00318             ahtdm, rhtdm, avtdm, rvtdm);
00319     for (iq = 0; iq < ctl.nq; iq++) {
00320       fprintf(out, " ");
00321       fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00322       fprintf(out, " ");
00323       fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00324     }
00325     fprintf(out, " %d\n", np);
00326   }
00327
00328   /* Close file... */
00329   fclose(out);
00330
00331   /* Free... */
00332   free(atm1);
00333   free(atm2);
00334   free(lon1_old);
00335   free(lat1_old);
00336   free(z1_old);
00337   free(lh1);
00338   free(lv1);
00339   free(lon2_old);
00340   free(lat2_old);
00341   free(z2_old);
00342   free(lh2);
00343   free(lv2);
00344   free(ahtd);
00345   free(avtd);
00346   free(aqtd);
00347   free(rhtd);
00348   free(rvtd);
00349   free(rqtd);
00350   free(work);
00351
00352   return EXIT_SUCCESS;
00353 }
```

5.5 atm_init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

- int main (int argc, char ∗argv[ ])

5.5.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file atm_init.c.

5.5.2 Function Documentation

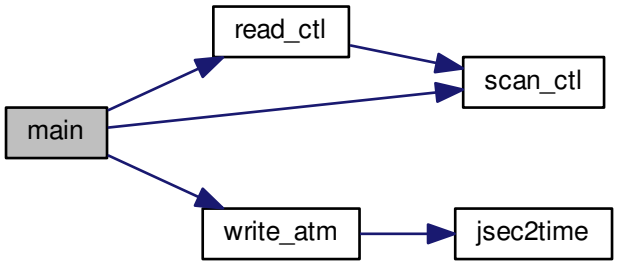
5.5.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 27 of file atm_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:



5.6 atm_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   Copyright (C) 2013-2019 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 }
```

5.7 atm_split.c File Reference

Split air parcels into a larger number of parcels.

Functions

- int main (int argc, char ∗argv[ ])

5.7.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file atm_split.c.

5.7.2 Function Documentation

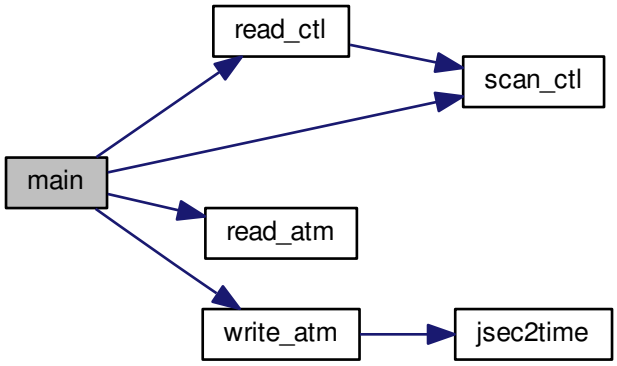
5.7.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 27 of file atm_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   if (!read_atm(argv[2], &ctl, atm))
00072     ERRMSG("Cannot open file!");
00073
00074   /* Get total and maximum mass... */
00075   if (ctl.qnt_m >= 0)
00076     for (ip = 0; ip < atm->np; ip++) {
```

```
00077       mtot += atm->q[ctl.qnt_m][ip];
00078       mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00079     }
00080   if (m > 0)
00081     mtot = m;
00082
00083   /* Loop over air parcels... */
00084   for (i = 0; i < n; i++) {
00085
00086     /* Select air parcel... */
00087     if (ctl.qnt_m >= 0)
00088       do {
00089         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00090       } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00091     else
00092       ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00093
00094     /* Set time... */
00095     if (t1 > t0)
00096       atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00097     else
00098       atm2->time[atm2->np] = atm->time[ip]
00099         + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00100
00101     /* Set vertical position... */
00102     if (z1 > z0)
00103       atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00104     else
00105       atm2->p[atm2->np] = atm->p[ip]
00106         + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00107
00108     /* Set horizontal position... */
00109     if (lon1 > lon0 && lat1 > lat0) {
00110       atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00111       atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00112     } else {
00113       atm2->lon[atm2->np] = atm->lon[ip]
00114         + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00115       atm2->lat[atm2->np] = atm->lat[ip]
00116         + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
00117     }
00118
00119     /* Copy quantities... */
00120     for (iq = 0; iq < ctl.nq; iq++)
00121       atm2->q[iq][atm2->np] = atm->q[iq][ip];
00122
00123     /* Adjust mass... */
00124     if (ctl.qnt_m >= 0)
00125       atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00126
00127     /* Increment particle counter... */
00128     if ((++atm2->np) >= NP)
00129       ERRMSG("Too many air parcels!");
00130   }
00131
00132   /* Save data and close file... */
00133   write_atm(argv[3], &ctl, atm2, atm->time[0]);
00134
00135   /* Free... */
00136   free(atm);
00137   free(atm2);
00138
00139   return EXIT_SUCCESS;
00140 }
```

Here is the call graph for this function:



5.8 atm_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) 2013-2019 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   if (!read_atm(argv[2], &ctl, atm))
00072     ERRMSG("Cannot open file!");
00073
00074   /* Get total and maximum mass... */
00075   if (ctl.qnt_m >= 0)
00076     for (ip = 0; ip < atm->np; ip++) {
00077       mtot += atm->q[ctl.qnt_m][ip];
00078       mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00079     }
00080   if (m > 0)
00081     mtot = m;
00082
00083   /* Loop over air parcels... */
00084   for (i = 0; i < n; i++) {
00085
00086     /* Select air parcel... */
00087     if (ctl.qnt_m >= 0)
00088       do {
00089         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00090       } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00091     else
00092       ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00093
00094     /* Set time... */
00095     if (t1 > t0)
00096       atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00097     else
00098       atm2->time[atm2->np] = atm->time[ip]
00099         + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00100
00101     /* Set vertical position... */
00102     if (z1 > z0)
00103       atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00104     else
00105       atm2->p[atm2->np] = atm->p[ip]
00106         + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00107
00108     /* Set horizontal position... */
00109     if (lon1 > lon0 && lat1 > lat0) {
00110       atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00111       atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00112     } else {
00113       atm2->lon[atm2->np] = atm->lon[ip]
00114         + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00115       atm2->lat[atm2->np] = atm->lat[ip]
00116         + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
00117     }
00118
00119     /* Copy quantities... */
00120     for (iq = 0; iq < ctl.nq; iq++)
00121       atm2->q[iq][atm2->np] = atm->q[iq][ip];
00122
00123     /* Adjust mass... */
00124     if (ctl.qnt_m >= 0)
00125       atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00126
00127     /* Increment particle counter... */
00128     if ((++atm2->np) >= NP)
00129       ERRMSG("Too many air parcels!");
00130   }
00131
00132   /* Save data and close file... */
00133   write_atm(argv[3], &ctl, atm2, atm->time[0]);
00134
00135   /* Free... */
00136   free(atm);
00137   free(atm2);
00138
00139   return EXIT_SUCCESS;
00140 }
```

5.9 atm_stat.c File Reference

Calculate air parcel statistics.

Functions

- int main (int argc, char ∗argv[ ])

5.9.1 Detailed Description

Calculate air parcel statistics.

Definition in file atm_stat.c.

5.9.2 Function Documentation

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

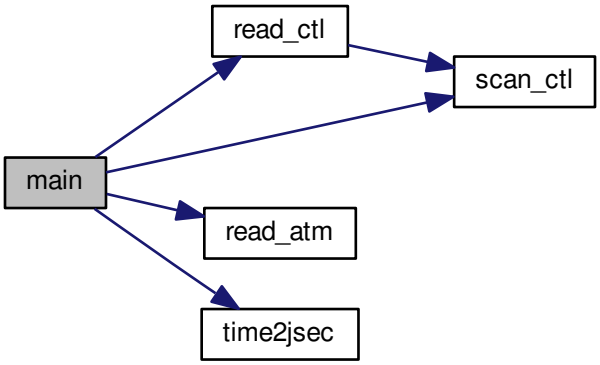
Definition at line 27 of file atm_stat.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm, *atm_filt;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040     t, t0, qm[NQ], *work, zm, *zs;
00041
00042   int ens, f, init = 0, ip, iq, year, mon, day, hour, min;
00043
00044   /* Allocate... */
00045   ALLOC(atm, atm_t, 1);
00046   ALLOC(atm_filt, atm_t, 1);
00047   ALLOC(work, double,
00048         NP);
00049   ALLOC(zs, double,
00050         NP);
00051
00052   /* Check arguments... */
00053   if (argc < 4)
00054     ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
00055
00056   /* Read control parameters... */
00057   read_ctl(argv[1], argc, argv, &ctl);
00058   ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059   p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060   p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061   lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062   lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063   lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064   lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066   /* Write info... */
00067   printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069   /* Create output file... */
00070   if (!(out = fopen(argv[2], "w")))
00071     ERRMSG("Cannot create file!");
00072
00073   /* Write header... */
00074   fprintf(out,
00075           "# $1 = time [s]\n"
00076           "# $2 = time difference [s]\n"
```

```
00077           "# $3 = altitude (%s) [km]\n"
00078           "# $4 = longitude (%s) [deg]\n"
00079           "# $5 = latitude (%s) [deg]\n", argv[3], argv[3], argv[3]);
00080   for (iq = 0; iq < ctl.nq; iq++)
00081     fprintf(out, "# $%d = %s (%s) [%s]\n", iq + 6,
00082             ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083   fprintf(out, "# $%d = number of particles\n\n", ctl.nq + 6);
00084
00085   /* Loop over files... */
00086   for (f = 4; f < argc; f++) {
00087
00088     /* Read atmopheric data... */
00089     if (!read_atm(argv[f], &ctl, atm))
00090       continue;
00091
00092     /* Get time from filename... */
00093     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094     year = atoi(tstr);
00095     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096     mon = atoi(tstr);
00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save intial time... */
00106     if (!init) {
00107       init = 1;
00108       t0 = t;
00109     }
00110
00111     /* Filter data... */
00112     atm_filt->np = 0;
00113     for (ip = 0; ip < atm->np; ip++) {
00114
00115       /* Check time... */
00116       if (!gsl_finite(atm->time[ip]))
00117         continue;
00118
00119       /* Check ensemble index... */
00120       if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00121         continue;
00122
00123       /* Check spatial range... */
00124       if (atm->p[ip] > p0 || atm->p[ip] < p1
00125           || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00126           || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00127         continue;
00128
00129       /* Save data... */
00130       atm_filt->time[atm_filt->np] = atm->time[ip];
00131       atm_filt->p[atm_filt->np] = atm->p[ip];
00132       atm_filt->lon[atm_filt->np] = atm->lon[ip];
00133       atm_filt->lat[atm_filt->np] = atm->lat[ip];
00134       for (iq = 0; iq < ctl.nq; iq++)
00135         atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00136       atm_filt->np++;
00137     }
00138
00139     /* Get heights... */
00140     for (ip = 0; ip < atm_filt->np; ip++)
00141       zs[ip] = Z(atm_filt->p[ip]);
00142
00143     /* Get statistics... */
00144     if (strcasecmp(argv[3], "mean") == 0) {
00145       zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00146       lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00147       latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00148       for (iq = 0; iq < ctl.nq; iq++)
00149         qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00150     } else if (strcasecmp(argv[3], "stddev") == 0) {
00151       zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00152       lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00153       latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00154       for (iq = 0; iq < ctl.nq; iq++)
00155         qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcasecmp(argv[3], "min") == 0) {
00157       zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00158       lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00159       latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00160       for (iq = 0; iq < ctl.nq; iq++)
00161         qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcasecmp(argv[3], "max") == 0) {
00163       zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
```

```
00164       lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00165       latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00166       for (iq = 0; iq < ctl.nq; iq++)
00167         qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcasecmp(argv[3], "skew") == 0) {
00169       zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00170       lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00171       latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00172       for (iq = 0; iq < ctl.nq; iq++)
00173         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcasecmp(argv[3], "kurt") == 0) {
00175       zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00176       lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00177       latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00178       for (iq = 0; iq < ctl.nq; iq++)
00179         qm[iq] =
00180           gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00181     } else if (strcasecmp(argv[3], "median") == 0) {
00182       zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00183       lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00184       latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00185       for (iq = 0; iq < ctl.nq; iq++)
00186         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187     } else if (strcasecmp(argv[3], "absdev") == 0) {
00188       zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00189       lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00190       latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00191       for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193     } else if (strcasecmp(argv[3], "mad") == 0) {
00194       zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00195       lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00196       latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00197       for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] =
00199           gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00200     } else
00201       ERRMSG("Unknown parameter!");
00202
00203     /* Write data... */
00204     fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00205     for (iq = 0; iq < ctl.nq; iq++) {
00206       fprintf(out, " ");
00207       fprintf(out, ctl.qnt_format[iq], qm[iq]);
00208     }
00209     fprintf(out, " %d\n", atm_filt->np);
00210   }
00211
00212   /* Close file... */
00213   fclose(out);
00214
00215   /* Free... */
00216   free(atm);
00217   free(atm_filt);
00218   free(work);
00219   free(zs);
00220
00221   return EXIT_SUCCESS;
00222 }
```

Here is the call graph for this function:



5.10 atm_stat.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) 2013-2019 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, *atm_filt;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040     t, t0, qm[NQ], *work, zm, *zs;
00041
00042   int ens, f, init = 0, ip, iq, year, mon, day, hour, min;
00043
00044   /* Allocate... */
00045   ALLOC(atm, atm_t, 1);
00046   ALLOC(atm_filt, atm_t, 1);
00047   ALLOC(work, double,
00048         NP);
00049   ALLOC(zs, double,
00050         NP);
00051
00052   /* Check arguments... */
00053   if (argc < 4)
00054     ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
```

```
00055
00056   /* Read control parameters... */
00057   read_ctl(argv[1], argc, argv, &ctl);
00058   ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059   p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060   p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061   lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062   lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063   lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064   lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066   /* Write info... */
00067   printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069   /* Create output file... */
00070   if (!(out = fopen(argv[2], "w")))
00071     ERRMSG("Cannot create file!");
00072
00073   /* Write header... */
00074   fprintf(out,
00075           "# $1 = time [s]\n"
00076           "# $2 = time difference [s]\n"
00077           "# $3 = altitude (%s) [km]\n"
00078           "# $4 = longitude (%s) [deg]\n"
00079           "# $5 = latitude (%s) [deg]\n", argv[3], argv[3], argv[3]);
00080   for (iq = 0; iq < ctl.nq; iq++)
00081     fprintf(out, "# $%d = %s (%s) [%s]\n", iq + 6,
00082             ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083   fprintf(out, "# $%d = number of particles\n\n", ctl.nq + 6);
00084
00085   /* Loop over files... */
00086   for (f = 4; f < argc; f++) {
00087
00088     /* Read atmopheric data... */
00089     if (!read_atm(argv[f], &ctl, atm))
00090       continue;
00091
00092     /* Get time from filename... */
00093     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094     year = atoi(tstr);
00095     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096     mon = atoi(tstr);
00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save intial time... */
00106     if (!init) {
00107       init = 1;
00108       t0 = t;
00109     }
00110
00111     /* Filter data... */
00112     atm_filt->np = 0;
00113     for (ip = 0; ip < atm->np; ip++) {
00114
00115       /* Check time... */
00116       if (!gsl_finite(atm->time[ip]))
00117         continue;
00118
00119       /* Check ensemble index... */
00120       if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00121         continue;
00122
00123       /* Check spatial range... */
00124       if (atm->p[ip] > p0 || atm->p[ip] < p1
00125           || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00126           || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00127         continue;
00128
00129       /* Save data... */
00130       atm_filt->time[atm_filt->np] = atm->time[ip];
00131       atm_filt->p[atm_filt->np] = atm->p[ip];
00132       atm_filt->lon[atm_filt->np] = atm->lon[ip];
00133       atm_filt->lat[atm_filt->np] = atm->lat[ip];
00134       for (iq = 0; iq < ctl.nq; iq++)
00135         atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00136       atm_filt->np++;
00137     }
00138
00139     /* Get heights... */
00140     for (ip = 0; ip < atm_filt->np; ip++)
00141       zs[ip] = Z(atm_filt->p[ip]);
```

```
00142
00143     /* Get statistics... */
00144     if (strcasecmp(argv[3], "mean") == 0) {
00145       zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00146       lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00147       latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00148       for (iq = 0; iq < ctl.nq; iq++)
00149         qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00150     } else if (strcasecmp(argv[3], "stddev") == 0) {
00151       zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00152       lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00153       latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00154       for (iq = 0; iq < ctl.nq; iq++)
00155         qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcasecmp(argv[3], "min") == 0) {
00157       zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00158       lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00159       latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00160       for (iq = 0; iq < ctl.nq; iq++)
00161         qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcasecmp(argv[3], "max") == 0) {
00163       zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00164       lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00165       latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00166       for (iq = 0; iq < ctl.nq; iq++)
00167         qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcasecmp(argv[3], "skew") == 0) {
00169       zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00170       lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00171       latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00172       for (iq = 0; iq < ctl.nq; iq++)
00173         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcasecmp(argv[3], "kurt") == 0) {
00175       zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00176       lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00177       latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00178       for (iq = 0; iq < ctl.nq; iq++)
00179         qm[iq] =
00180           gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00181     } else if (strcasecmp(argv[3], "median") == 0) {
00182       zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00183       lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00184       latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00185       for (iq = 0; iq < ctl.nq; iq++)
00186         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187     } else if (strcasecmp(argv[3], "absdev") == 0) {
00188       zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00189       lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00190       latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00191       for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193     } else if (strcasecmp(argv[3], "mad") == 0) {
00194       zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00195       lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00196       latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00197       for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] =
00199           gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00200     } else
00201       ERRMSG("Unknown parameter!");
00202
00203     /* Write data... */
00204     fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00205     for (iq = 0; iq < ctl.nq; iq++) {
00206       fprintf(out, " ");
00207       fprintf(out, ctl.qnt_format[iq], qm[iq]);
00208     }
00209     fprintf(out, " %d\n", atm_filt->np);
00210   }
00211
00212   /* Close file... */
00213   fclose(out);
00214
00215   /* Free... */
00216   free(atm);
00217   free(atm_filt);
00218   free(work);
00219   free(zs);
00220
00221   return EXIT_SUCCESS;
00222 }
```

5.11 day2doy.c File Reference

Convert date to day of year.

Functions

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

5.11.1 Detailed Description

Convert date to day of year.

Definition in file day2doy.c.

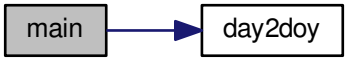
5.11.2 Function Documentation

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

Definition at line 27 of file day2doy.c.

```
00029                 {
00030
00031   int day, doy, mon, year;
00032
00033   /* Check arguments... */
00034   if (argc < 4)
00035     ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037   /* Read arguments... */
00038   year = atoi(argv[1]);
00039   mon = atoi(argv[2]);
00040   day = atoi(argv[3]);
00041
00042   /* Convert... */
00043   day2doy(year, mon, day, &doy);
00044   printf("%d %d\n", year, doy);
00045
00046   return EXIT_SUCCESS;
00047 }
```

Here is the call graph for this function:



5.12 day2doy.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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   int day, doy, mon, year;
00032
00033   /* Check arguments... */
00034   if (argc < 4)
00035     ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037   /* Read arguments... */
00038   year = atoi(argv[1]);
00039   mon = atoi(argv[2]);
00040   day = atoi(argv[3]);
00041
00042   /* Convert... */
00043   day2doy(year, mon, day, &doy);
00044   printf("%d %d\n", year, doy);
00045
00046   return EXIT_SUCCESS;
00047 }
```

5.13 doy2day.c File Reference

Convert day of year to date.

Functions

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

5.13.1 Detailed Description

Convert day of year to date.

Definition in file doy2day.c.

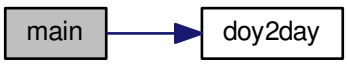
5.13.2 Function Documentation

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

Definition at line 27 of file doy2day.c.

```
00029                {
00030
00031   int day, doy, mon, year;
00032
00033   /* Check arguments... */
00034   if (argc < 3)
00035     ERRMSG("Give parameters: <year> <doy>");
00036
00037   /* Read arguments... */
00038   year = atoi(argv[1]);
00039   doy = atoi(argv[2]);
00040
00041   /* Convert... */
00042   doy2day(year, doy, &mon, &day);
00043   printf("%d %d %d\n", year, mon, day);
00044
00045   return EXIT_SUCCESS;
00046 }
```

Here is the call graph for this function:



5.14 doy2day.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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   int day, doy, mon, year;
00032
00033   /* Check arguments... */
00034   if (argc < 3)
00035     ERRMSG("Give parameters: <year> <doy>");
00036
```

```
00037   /* Read arguments... */
00038   year = atoi(argv[1]);
00039   doy = atoi(argv[2]);
00040
00041   /* Convert... */
00042   doy2day(year, doy, &mon, &day);
00043   printf("%d %d %d\n", year, mon, day);
00044
00045   return EXIT_SUCCESS;
00046 }
```

5.15 extract.c File Reference

Extract single trajectory from atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.15.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

5.15.2 Function Documentation

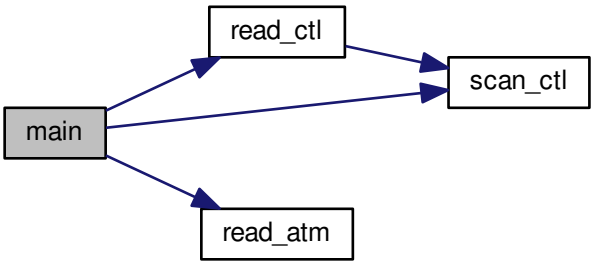
5.15.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 27 of file extract.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   FILE *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> <trajec.tab> <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, "", 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 (!read_atm(argv[f], &ctl, atm))
00072       continue;
00073
00074     /* Check air parcel index... */
00075     if (ip > atm->np)
00076       ERRMSG("Air parcel index out of range!");
00077
00078     /* Write data... */
00079     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081     for (iq = 0; iq < ctl.nq; iq++) {
00082       fprintf(out, " ");
00083       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084     }
00085     fprintf(out, "\n");
00086   }
00087
00088   /* Close file... */
00089   fclose(out);
00090
00091   /* Free... */
00092   free(atm);
00093
00094   return EXIT_SUCCESS;
00095 }
```

Here is the call graph for this function:



5.16 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) 2013-2019 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   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> <trajec.tab> <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, "", 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 (!read_atm(argv[f], &ctl, atm))
00072       continue;
00073
00074     /* Check air parcel index... */
00075     if (ip > atm->np)
00076       ERRMSG("Air parcel index out of range!");
00077
00078     /* Write data... */
00079     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081     for (iq = 0; iq < ctl.nq; iq++) {
00082       fprintf(out, " ");
00083       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084     }
00085     fprintf(out, "\n");
00086   }
00087
00088   /* Close file... */
00089   fclose(out);
00090
00091   /* Free... */
00092   free(atm);
00093
00094   return EXIT_SUCCESS;
00095 }
```

5.17 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

- int main (int argc, char ∗argv[ ])

5.17.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

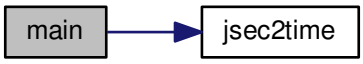
5.17.2 Function Documentation

5.17.2.1 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:



5.18 jsec2time.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 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 }
```

5.19 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.
- void day2doy (int year, int mon, int day, int *doy)

  Get day of year from date.
- void doy2day (int year, int doy, int *mon, int *day)

  Get date from day of year.
- 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 get_met_replace (char *orig, char *search, char *repl)

  Replace template strings in filename.
- 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 *pv, 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 *pv, 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_irr (double *xx, int n, double x)

  Find array index for irregular grid.
- int locate_reg (double *xx, int n, double x)

  Find array index for regular grid.
- int 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.
- int 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_pv (met_t *met)

  Calculate potential vorticity.
- 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.

5.19.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

5.19.2 Function Documentation

5.19.2.1 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 }
```

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

Climatology of HNO3 volume mixing ratios.

Definition at line 45 of file libtrac.c.

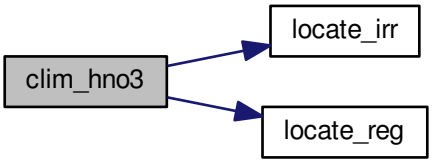
```
00048            {
00049
00050   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051     9072000.00, 11664000.00, 14342400.00,
00052     16934400.00, 19612800.00, 22291200.00,
00053     24883200.00, 27561600.00, 30153600.00
00054   };
00055
00056   static double lats[18] = { -85, -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    isec = locate_irr(secs, 12, sec);
00292    ilat = locate_reg(lats, 18, lat);
00293    ip = locate_irr(ps, 10, p);
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:



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

Climatology of tropopause pressure.

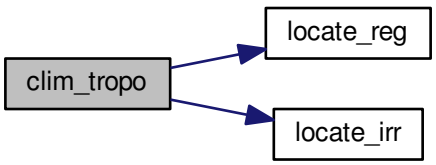
Definition at line 311 of file libtrac.c.

```
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;
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   ilat = locate_reg(lats, 73, lat);
00439   imon = locate_irr(doys, 12, doy);
00440
00441   /* Interpolate... */
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   return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447 }
```

Here is the call graph for this function:



5.19.2.4 void day2doy (int *year*, int *mon*, int *day*, int * *doy*)

Get day of year from date.

Definition at line 451 of file libtrac.c.

```
00455            {
00456
00457   int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00458   int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00459
00460   /* Get day of year... */
00461   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00462     *doy = d01[mon - 1] + day - 1;
00463   else
00464     *doy = d0[mon - 1] + day - 1;
00465 }
```

5.19.2.5 void doy2day (int *year*, int *doy*, int * *mon*, int * *day*)

Get date from day of year.

Definition at line 469 of file libtrac.c.

```
00473            {
00474
00475   int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00476   int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00477   int i;
00478
00479   /* Get month and day... */
00480   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00481     for (i = 11; i >= 0; i--)
00482       if (d0l[i] <= doy)
00483         break;
00484     *mon = i + 1;
00485     *day = doy - d0l[i] + 1;
00486   } else {
00487     for (i = 11; i >= 0; i--)
00488       if (d0[i] <= doy)
00489         break;
00490     *mon = i + 1;
00491     *day = doy - d0[i] + 1;
00492   }
00493 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 497 of file libtrac.c.

```
00501             {
00502
00503   double radius;
00504
00505   radius = z + RE;
00506   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00507   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00508   x[2] = radius * sin(lat / 180 * M_PI);
00509 }
```

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

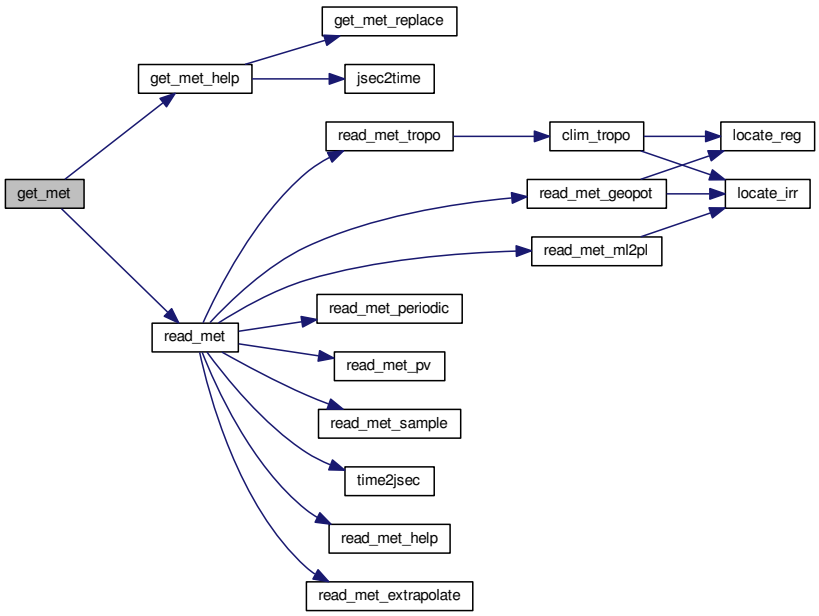
Get meteorological data for given timestep.

Definition at line 513 of file libtrac.c.

```
00518                 {
00519
00520   static int init, ip, ix, iy;
00521
00522   met_t *mets;
00523
00524   char filename[LEN];
00525
00526   /* Init... */
00527   if (t == ctl->t_start || !init) {
00528     init = 1;
00529
00530     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00531     if (!read_met(ctl, filename, *met0))
00532       ERRMSG("Cannot open file!");
00533
00534     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00535     if (!read_met(ctl, filename, *met1))
00536       ERRMSG("Cannot open file!");
```

```
00537   }
00538
00539   /* Read new data for forward trajectories... */
00540   if (t > (*met1)->time && ctl->direction == 1) {
00541     mets = *met1;
00542     *met1 = *met0;
00543     *met0 = mets;
00544     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00545     if (!read_met(ctl, filename, *met1))
00546       ERRMSG("Cannot open file!");
00547   }
00548
00549   /* Read new data for backward trajectories... */
00550   if (t < (*met0)->time && ctl->direction == -1) {
00551     mets = *met1;
00552     *met1 = *met0;
00553     *met0 = mets;
00554     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00555     if (!read_met(ctl, filename, *met0))
00556       ERRMSG("Cannot open file!");
00557   }
00558
00559   /* Check that grids are consistent... */
00560   if ((*met0)->nx != (*met1)->nx
00561       || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00562     ERRMSG("Meteo grid dimensions do not match!");
00563   for (ix = 0; ix < (*met0)->nx; ix++)
00564     if ((*met0)->lon[ix] != (*met1)->lon[ix])
00565       ERRMSG("Meteo grid longitudes do not match!");
00566   for (iy = 0; iy < (*met0)->ny; iy++)
00567     if ((*met0)->lat[iy] != (*met1)->lat[iy])
00568       ERRMSG("Meteo grid latitudes do not match!");
00569   for (ip = 0; ip < (*met0)->np; ip++)
00570     if ((*met0)->p[ip] != (*met1)->p[ip])
00571       ERRMSG("Meteo grid pressure levels do not match!");
00572 }
```

Here is the call graph for this function:



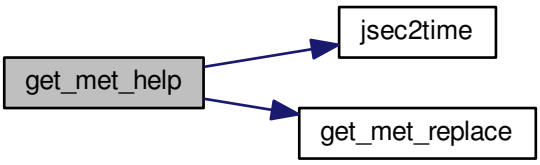
5.19.2.8 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

Get meteorological data for timestep.

Definition at line 576 of file libtrac.c.

```
00581                   {
00582
00583   char repl[LEN];
00584
00585   double t6, r;
00586
00587   int year, mon, day, hour, min, sec;
00588
00589   /* Round time to fixed intervals... */
00590   if (direct == -1)
00591     t6 = floor(t / dt_met) * dt_met;
00592   else
00593     t6 = ceil(t / dt_met) * dt_met;
00594
00595   /* Decode time... */
00596   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00597
00598   /* Set filename... */
00599   sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
00600   sprintf(repl, "%d", year);
00601   get_met_replace(filename, "YYYY", repl);
00602   sprintf(repl, "%02d", mon);
00603   get_met_replace(filename, "MM", repl);
00604   sprintf(repl, "%02d", day);
00605   get_met_replace(filename, "DD", repl);
00606   sprintf(repl, "%02d", hour);
00607   get_met_replace(filename, "HH", repl);
00608 }
```

Here is the call graph for this function:



5.19.2.9 void get_met_replace (char * *orig*, char * *search*, char * *repl*)

Replace template strings in filename.

Definition at line 612 of file libtrac.c.

```
00615               {
00616
00617   char buffer[LEN], *ch;
00618
00619   int i;
00620
00621   /* Iterate... */
00622   for (i = 0; i < 3; i++) {
00623
00624     /* Replace substring... */
00625     if (!(ch = strstr(orig, search)))
00626       return;
00627     strncpy(buffer, orig, (size_t) (ch - orig));
00628     buffer[ch - orig] = 0;
00629     sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
00630     orig[0] = 0;
00631     strcpy(orig, buffer);
00632   }
00633 }
```

5.19.2.10 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 637 of file libtrac.c.

```
00643                {
00644
00645   double aux00, aux01, aux10, aux11;
00646
00647   /* Set variables... */
00648   aux00 = array[ix][iy];
00649   aux01 = array[ix][iy + 1];
00650   aux10 = array[ix + 1][iy];
00651   aux11 = array[ix + 1][iy + 1];
00652
00653   /* Interpolate horizontally... */
00654   aux00 = wy * (aux00 - aux01) + aux01;
00655   aux11 = wy * (aux10 - aux11) + aux11;
00656   *var = wx * (aux00 - aux11) + aux11;
00657 }
```

5.19.2.11 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 661 of file libtrac.c.

```
00669                {
00670
00671   double aux00, aux01, aux10, aux11;
00672
00673   /* Interpolate vertically... */
00674   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00675     + array[ix][iy][ip + 1];
00676   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00677     + array[ix][iy + 1][ip + 1];
00678   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00679     + array[ix + 1][iy][ip + 1];
00680   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00681     + array[ix + 1][iy + 1][ip + 1];
00682
00683   /* Interpolate horizontally... */
00684   aux00 = wy * (aux00 - aux01) + aux01;
00685   aux11 = wy * (aux10 - aux11) + aux11;
00686   *var = wx * (aux00 - aux11) + aux11;
00687 }
```

5.19.2.12 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 * *pv*, double * *h2o*, double * *o3*)

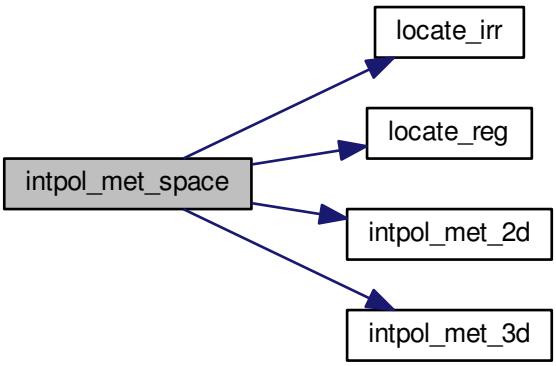
Spatial interpolation of meteorological data.

Definition at line 691 of file libtrac.c.

```
00705                {
00706
00707   double wp, wx, wy;
00708
00709   int ip, ix, iy;
00710
00711   /* Check longitude... */
00712   if (met->lon[met->nx - 1] > 180 && lon < 0)
00713     lon += 360;
00714
00715   /* Get indices... */
00716   ip = locate_irr(met->p, met->np, p);
00717   ix = locate_reg(met->lon, met->nx, lon);
00718   iy = locate_reg(met->lat, met->ny, lat);
```

```
00719
00720   /* Get weights... */
00721   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00722   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00723   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00724
00725   /* Interpolate... */
00726   if (ps != NULL)
00727     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00728   if (pt != NULL)
00729     intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00730   if (z != NULL)
00731     intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00732   if (t != NULL)
00733     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00734   if (u != NULL)
00735     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00736   if (v != NULL)
00737     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00738   if (w != NULL)
00739     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00740   if (pv != NULL)
00741     intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00742   if (h2o != NULL)
00743     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00744   if (o3 != NULL)
00745     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00746 }
```

Here is the call graph for this function:



5.19.2.13 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 * *pv*, double * *h2o*, double * *o3*)

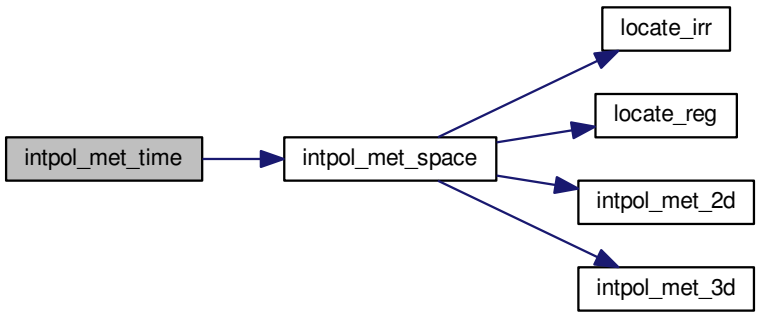
Temporal interpolation of meteorological data.

Definition at line 750 of file libtrac.c.

```
00766               {
00767
00768   double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00769     v0, v1, w0, w1, wt, z0, z1;
00770
00771   /* Spatial interpolation... */
00772   intpol_met_space(met0, p, lon, lat,
00773                    ps == NULL ? NULL : &ps0,
00774                    pt == NULL ? NULL : &pt0,
00775                    z == NULL ? NULL : &z0,
```

```
00776                   t == NULL ? NULL : &t0,
00777                   u == NULL ? NULL : &u0,
00778                   v == NULL ? NULL : &v0,
00779                   w == NULL ? NULL : &w0,
00780                   pv == NULL ? NULL : &pv0,
00781                   h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00782   intpol_met_space(met1, p, lon, lat,
00783                   ps == NULL ? NULL : &ps1,
00784                   pt == NULL ? NULL : &pt1,
00785                   z == NULL ? NULL : &z1,
00786                   t == NULL ? NULL : &t1,
00787                   u == NULL ? NULL : &u1,
00788                   v == NULL ? NULL : &v1,
00789                   w == NULL ? NULL : &w1,
00790                   pv == NULL ? NULL : &pv1,
00791                   h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00792
00793   /* Get weighting factor... */
00794   wt = (met1->time - ts) / (met1->time - met0->time);
00795
00796   /* Interpolate... */
00797   if (ps != NULL)
00798     *ps = wt * (ps0 - ps1) + ps1;
00799   if (pt != NULL)
00800     *pt = wt * (pt0 - pt1) + pt1;
00801   if (z != NULL)
00802     *z = wt * (z0 - z1) + z1;
00803   if (t != NULL)
00804     *t = wt * (t0 - t1) + t1;
00805   if (u != NULL)
00806     *u = wt * (u0 - u1) + u1;
00807   if (v != NULL)
00808     *v = wt * (v0 - v1) + v1;
00809   if (w != NULL)
00810     *w = wt * (w0 - w1) + w1;
00811   if (pv != NULL)
00812     *pv = wt * (pv0 - pv1) + pv1;
00813   if (h2o != NULL)
00814     *h2o = wt * (h2o0 - h2o1) + h2o1;
00815   if (o3 != NULL)
00816     *o3 = wt * (o30 - o31) + o31;
00817 }
```

Here is the call graph for this function:



5.19.2.14 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 821 of file libtrac.c.

```
00829                   {
00830
00831   struct tm t0, *t1;
00832
00833   time_t jsec0;
00834
00835   t0.tm_year = 100;
00836   t0.tm_mon = 0;
00837   t0.tm_mday = 1;
00838   t0.tm_hour = 0;
00839   t0.tm_min = 0;
00840   t0.tm_sec = 0;
00841
00842   jsec0 = (time_t) jsec + timegm(&t0);
00843   t1 = gmtime(&jsec0);
00844
00845   *year = t1->tm_year + 1900;
00846   *mon = t1->tm_mon + 1;
00847   *day = t1->tm_mday;
00848   *hour = t1->tm_hour;
00849   *min = t1->tm_min;
00850   *sec = t1->tm_sec;
00851   *remain = jsec - floor(jsec);
00852 }
```

5.19.2.15 int locate_irr (double * *xx*, int *n*, double *x*)

Find array index for irregular grid.

Definition at line 856 of file libtrac.c.

```
00859             {
00860
00861   int i, ilo, ihi;
00862
00863   ilo = 0;
00864   ihi = n - 1;
00865   i = (ihi + ilo) >> 1;
00866
00867   if (xx[i] < xx[i + 1])
00868     while (ihi > ilo + 1) {
00869       i = (ihi + ilo) >> 1;
00870       if (xx[i] > x)
00871         ihi = i;
00872       else
00873         ilo = i;
00874   } else
00875     while (ihi > ilo + 1) {
00876       i = (ihi + ilo) >> 1;
00877       if (xx[i] <= x)
00878         ihi = i;
00879       else
00880         ilo = i;
00881     }
00882
00883   return ilo;
00884 }
```

5.19.2.16 int locate_reg (double * *xx*, int *n*, double *x*)

Find array index for regular grid.

Definition at line 888 of file libtrac.c.

```
00891             {
00892
00893   int i;
00894
00895   /* Calculate index... */
00896   i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00897
00898   /* Check range... */
00899   if (i < 0)
00900     i = 0;
00901   else if (i >= n - 2)
00902     i = n - 2;
00903
00904   return i;
00905 }
```

5.19.2.17 int read_atm (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Read atmospheric data.

Definition at line 909 of file libtrac.c.

```
00912                {
00913
00914   FILE *in;
00915
00916   char line[LEN], *tok;
00917
00918   double t0;
00919
00920   int dimid, ip, iq, ncid, varid;
00921
00922   size_t nparts;
00923
00924   /* Init... */
00925   atm->np = 0;
00926
00927   /* Write info... */
00928   printf("Read atmospheric data: %s\n", filename);
00929
00930   /* Read ASCII data... */
00931   if (ctl->atm_type == 0) {
00932
00933     /* Open file... */
00934     if (!(in = fopen(filename, "r")))
00935       return 0;
00936
00937     /* Read line... */
00938     while (fgets(line, LEN, in)) {
00939
00940       /* Read data... */
00941       TOK(line, tok, "%lg", atm->time[atm->np]);
00942       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00943       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00944       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00945       for (iq = 0; iq < ctl->nq; iq++)
00946         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00947
00948       /* Convert altitude to pressure... */
00949       atm->p[atm->np] = P(atm->p[atm->np]);
00950
00951       /* Increment data point counter... */
00952       if ((++atm->np) > NP)
00953         ERRMSG("Too many data points!");
00954     }
00955
00956     /* Close file... */
00957     fclose(in);
00958   }
00959
00960   /* Read binary data... */
00961   else if (ctl->atm_type == 1) {
00962
00963     /* Open file... */
00964     if (!(in = fopen(filename, "r")))
00965       return 0;
00966
00967     /* Read data... */
00968     FREAD(&atm->np, int, 1, in);
00969     FREAD(atm->time, double,
00970           (size_t) atm->np,
00971           in);
00972     FREAD(atm->p, double,
00973           (size_t) atm->np,
00974           in);
00975     FREAD(atm->lon, double,
00976           (size_t) atm->np,
00977           in);
00978     FREAD(atm->lat, double,
00979           (size_t) atm->np,
00980           in);
00981     for (iq = 0; iq < ctl->nq; iq++)
00982       FREAD(atm->q[iq], double,
00983             (size_t) atm->np,
00984             in);
00985
00986     /* Close file... */
00987     fclose(in);
00988   }
```

```
00989
00990   /* Read netCDF data... */
00991   else if (ctl->atm_type == 2) {
00992
00993     /* Open file... */
00994     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
00995       return 0;
00996
00997     /* Get dimensions... */
00998     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00999     NC(nc_inq_dimlen(ncid, dimid, &nparts));
01000     atm->np = (int) nparts;
01001     if (atm->np > NP)
01002       ERRMSG("Too many particles!");
01003
01004     /* Get time... */
01005     NC(nc_inq_varid(ncid, "time", &varid));
01006     NC(nc_get_var_double(ncid, varid, &t0));
01007     for (ip = 0; ip < atm->np; ip++)
01008       atm->time[ip] = t0;
01009
01010     /* Read geolocations... */
01011     NC(nc_inq_varid(ncid, "PRESS", &varid));
01012     NC(nc_get_var_double(ncid, varid, atm->p));
01013     NC(nc_inq_varid(ncid, "LON", &varid));
01014     NC(nc_get_var_double(ncid, varid, atm->lon));
01015     NC(nc_inq_varid(ncid, "LAT", &varid));
01016     NC(nc_get_var_double(ncid, varid, atm->lat));
01017
01018     /* Read variables... */
01019     if (ctl->qnt_p >= 0)
01020       if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01021         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01022     if (ctl->qnt_t >= 0)
01023       if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01024         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01025     if (ctl->qnt_u >= 0)
01026       if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01027         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
01028     if (ctl->qnt_v >= 0)
01029       if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01030         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01031     if (ctl->qnt_w >= 0)
01032       if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01033         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01034     if (ctl->qnt_h2o >= 0)
01035       if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01036         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01037     if (ctl->qnt_o3 >= 0)
01038       if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
01039         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01040     if (ctl->qnt_theta >= 0)
01041       if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01042         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01043     if (ctl->qnt_pv >= 0)
01044       if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01045         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01046
01047     /* Check data... */
01048     for (ip = 0; ip < atm->np; ip++)
01049       if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01050           || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01051           || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01052           || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01053           || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01054         atm->time[ip] = GSL_NAN;
01055         atm->p[ip] = GSL_NAN;
01056         atm->lon[ip] = GSL_NAN;
01057         atm->lat[ip] = GSL_NAN;
01058         for (iq = 0; iq < ctl->nq; iq++)
01059           atm->q[iq][ip] = GSL_NAN;
01060       } else {
01061         if (ctl->qnt_h2o >= 0)
01062           atm->q[ctl->qnt_h2o][ip] *= 1.608;
01063         if (ctl->qnt_pv >= 0)
01064           atm->q[ctl->qnt_pv][ip] *= 1e6;
01065         if (atm->lon[ip] > 180)
01066           atm->lon[ip] -= 360;
01067       }
01068
01069     /* Close file... */
01070     NC(nc_close(ncid));
01071   }
01072
01073   /* Error... */
01074   else
01075     ERRMSG("Atmospheric data type not supported!");
```

```
01076
01077   /* Check number of points... */
01078   if (atm->np < 1)
01079     ERRMSG("Can not read any data!");
01080
01081   /* Return success... */
01082   return 1;
01083 }
```

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

Read control parameters.

Definition at line 1087 of file libtrac.c.

```
01091               {
01092
01093   int ip, iq;
01094
01095   /* Write info... */
01096   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01097          "(executable: %s | compiled: %s, %s)\n\n",
01098          argv[0], __DATE__, __TIME__);
01099
01100   /* Initialize quantity indices... */
01101   ctl->qnt_ens = -1;
01102   ctl->qnt_m = -1;
01103   ctl->qnt_r = -1;
01104   ctl->qnt_rho = -1;
01105   ctl->qnt_ps = -1;
01106   ctl->qnt_pt = -1;
01107   ctl->qnt_z = -1;
01108   ctl->qnt_p = -1;
01109   ctl->qnt_t = -1;
01110   ctl->qnt_u = -1;
01111   ctl->qnt_v = -1;
01112   ctl->qnt_w = -1;
01113   ctl->qnt_h2o = -1;
01114   ctl->qnt_o3 = -1;
01115   ctl->qnt_theta = -1;
01116   ctl->qnt_vh = -1;
01117   ctl->qnt_vz = -1;
01118   ctl->qnt_pv = -1;
01119   ctl->qnt_tice = -1;
01120   ctl->qnt_tsts = -1;
01121   ctl->qnt_tnat = -1;
01122   ctl->qnt_stat = -1;
01123
01124   /* Read quantities... */
01125   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01126   if (ctl->nq > NQ)
01127     ERRMSG("Too many quantities!");
01128   for (iq = 0; iq < ctl->nq; iq++) {
01129
01130     /* Read quantity name and format... */
01131     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01132     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01133              ctl->qnt_format[iq]);
01134
01135     /* Try to identify quantity... */
01136     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01137       ctl->qnt_ens = iq;
01138       sprintf(ctl->qnt_unit[iq], "-");
01139     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01140       ctl->qnt_m = iq;
01141       sprintf(ctl->qnt_unit[iq], "kg");
01142     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01143       ctl->qnt_r = iq;
01144       sprintf(ctl->qnt_unit[iq], "m");
01145     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01146       ctl->qnt_rho = iq;
01147       sprintf(ctl->qnt_unit[iq], "kg/m^3");
01148     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01149       ctl->qnt_ps = iq;
01150       sprintf(ctl->qnt_unit[iq], "hPa");
01151     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01152       ctl->qnt_pt = iq;
01153       sprintf(ctl->qnt_unit[iq], "hPa");
01154     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01155       ctl->qnt_z = iq;
```

```
01156       sprintf(ctl->qnt_unit[iq], "km");
01157     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01158       ctl->qnt_p = iq;
01159       sprintf(ctl->qnt_unit[iq], "hPa");
01160     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01161       ctl->qnt_t = iq;
01162       sprintf(ctl->qnt_unit[iq], "K");
01163     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01164       ctl->qnt_u = iq;
01165       sprintf(ctl->qnt_unit[iq], "m/s");
01166     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01167       ctl->qnt_v = iq;
01168       sprintf(ctl->qnt_unit[iq], "m/s");
01169     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01170       ctl->qnt_w = iq;
01171       sprintf(ctl->qnt_unit[iq], "hPa/s");
01172     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01173       ctl->qnt_h2o = iq;
01174       sprintf(ctl->qnt_unit[iq], "1");
01175     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01176       ctl->qnt_o3 = iq;
01177       sprintf(ctl->qnt_unit[iq], "1");
01178     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01179       ctl->qnt_theta = iq;
01180       sprintf(ctl->qnt_unit[iq], "K");
01181     } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01182       ctl->qnt_vh = iq;
01183       sprintf(ctl->qnt_unit[iq], "m/s");
01184     } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01185       ctl->qnt_vz = iq;
01186       sprintf(ctl->qnt_unit[iq], "m/s");
01187     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01188       ctl->qnt_pv = iq;
01189       sprintf(ctl->qnt_unit[iq], "PVU");
01190     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01191       ctl->qnt_tice = iq;
01192       sprintf(ctl->qnt_unit[iq], "K");
01193     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01194       ctl->qnt_tsts = iq;
01195       sprintf(ctl->qnt_unit[iq], "K");
01196     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01197       ctl->qnt_tnat = iq;
01198       sprintf(ctl->qnt_unit[iq], "K");
01199     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01200       ctl->qnt_stat = iq;
01201       sprintf(ctl->qnt_unit[iq], "-");
01202     } else
01203       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01204   }
01205
01206   /* Time steps of simulation... */
01207   ctl->direction =
01208     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01209   if (ctl->direction != -1 && ctl->direction != 1)
01210     ERRMSG("Set DIRECTION to -1 or 1!");
01211   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
01212   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01213
01214   /* Meteorological data... */
01215   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01216   ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01217   ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01218   ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01219   ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01220   ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01221   ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01222   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01223   if (ctl->met_np > EP)
01224     ERRMSG("Too many levels!");
01225   for (ip = 0; ip < ctl->met_np; ip++)
01226     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01227   ctl->met_tropo
01228     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01229   scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01230   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01231   ctl->met_dt_out =
01232     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01233
01234   /* Isosurface parameters... */
01235   ctl->isosurf
01236     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01237   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01238
01239   /* Diffusion parameters... */
01240   ctl->turb_dx_trop
01241     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01242   ctl->turb_dx_strat
```

```
01243     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01244   ctl->turb_dz_trop
01245     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01246   ctl->turb_dz_strat
01247     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01248   ctl->turb_mesox =
01249     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01250   ctl->turb_mesoz =
01251     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01252
01253   /* Mass and life time... */
01254   ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01255   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01256   ctl->tdec_strat =
01257     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01258
01259   /* PSC analysis... */
01260   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01261   ctl->psc_hno3 =
01262     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01263
01264   /* Output of atmospheric data... */
01265   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
01266   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01267   ctl->atm_dt_out =
01268     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01269   ctl->atm_filter =
01270     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01271   ctl->atm_type =
01272     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01273
01274   /* Output of CSI data... */
01275   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01276   ctl->csi_dt_out =
01277     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01278   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
      csi_obsfile);
01279   ctl->csi_obsmin =
01280     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01281   ctl->csi_modmin =
01282     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01283   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01284   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01285   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01286   ctl->csi_lon0 =
01287     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01288   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01289   ctl->csi_nx =
01290     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01291   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01292   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01293   ctl->csi_ny =
01294     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01295
01296   /* Output of ensemble data... */
01297   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01298
01299   /* Output of grid data... */
01300   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01301            ctl->grid_basename);
01302   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01303   ctl->grid_dt_out =
01304     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01305   ctl->grid_sparse =
01306     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01307   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01308   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01309   ctl->grid_nz =
01310     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01311   ctl->grid_lon0 =
01312     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01313   ctl->grid_lon1 =
01314     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01315   ctl->grid_nx =
01316     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01317   ctl->grid_lat0 =
01318     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01319   ctl->grid_lat1 =
01320     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01321   ctl->grid_ny =
01322     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01323
01324   /* Output of profile data... */
```

```
01325   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01326            ctl->prof_basename);
01327   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
01328   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01329   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01330   ctl->prof_nz =
01331     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01332   ctl->prof_lon0 =
01333     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01334   ctl->prof_lon1 =
01335     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01336   ctl->prof_nx =
01337     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01338   ctl->prof_lat0 =
01339     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01340   ctl->prof_lat1 =
01341     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01342   ctl->prof_ny =
01343     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01344
01345   /* Output of station data... */
01346   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01347            ctl->stat_basename);
01348   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01349   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01350   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01351 }
```

Here is the call graph for this function:

5.19.2.19 int read_met (ctl_t * *ctl*, char * *filename*, met_t * *met*)

Read meteorological data file.

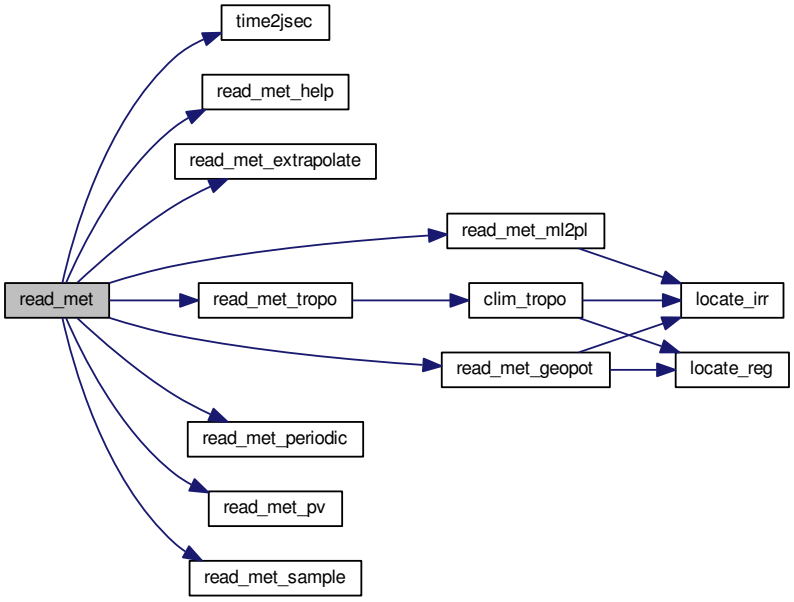
Definition at line 1355 of file libtrac.c.

```
01358               {
01359
01360   char cmd[2 * LEN], levname[LEN], tstr[10];
01361
01362   float help[EX * EY];
01363
01364   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01365
01366   size_t np, nx, ny;
01367
01368   /* Write info... */
01369   printf("Read meteorological data: %s\n", filename);
01370
01371   /* Open netCDF file... */
01372   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01373
01374     /* Try to stage meteo file... */
01375     if (ctl->met_stage[0] != '-') {
01376       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01377               year, mon, day, hour, filename);
01378       if (system(cmd) != 0)
01379         ERRMSG("Error while staging meteo data!");
01380     }
01381
01382     /* Try to open again... */
```

```
01383     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
01384       return 0;
01385   }
01386
01387   /* Get time from filename... */
01388   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01389   year = atoi(tstr);
01390   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01391   mon = atoi(tstr);
01392   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01393   day = atoi(tstr);
01394   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01395   hour = atoi(tstr);
01396   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01397
01398   /* Get dimensions... */
01399   NC(nc_inq_dimid(ncid, "lon", &dimid));
01400   NC(nc_inq_dimlen(ncid, dimid, &nx));
01401   if (nx < 2 || nx > EX)
01402     ERRMSG("Number of longitudes out of range!");
01403
01404   NC(nc_inq_dimid(ncid, "lat", &dimid));
01405   NC(nc_inq_dimlen(ncid, dimid, &ny));
01406   if (ny < 2 || ny > EY)
01407     ERRMSG("Number of latitudes out of range!");
01408
01409   sprintf(levname, "lev");
01410   NC(nc_inq_dimid(ncid, levname, &dimid));
01411   NC(nc_inq_dimlen(ncid, dimid, &np));
01412   if (np == 1) {
01413     sprintf(levname, "lev_2");
01414     NC(nc_inq_dimid(ncid, levname, &dimid));
01415     NC(nc_inq_dimlen(ncid, dimid, &np));
01416   }
01417   if (np < 2 || np > EP)
01418     ERRMSG("Number of levels out of range!");
01419
01420   /* Store dimensions... */
01421   met->np = (int) np;
01422   met->nx = (int) nx;
01423   met->ny = (int) ny;
01424
01425   /* Get horizontal grid... */
01426   NC(nc_inq_varid(ncid, "lon", &varid));
01427   NC(nc_get_var_double(ncid, varid, met->lon));
01428   NC(nc_inq_varid(ncid, "lat", &varid));
01429   NC(nc_get_var_double(ncid, varid, met->lat));
01430
01431   /* Read meteorological data... */
01432   read_met_help(ncid, "t", "T", met, met->t, 1.0);
01433   read_met_help(ncid, "u", "U", met, met->u, 1.0);
01434   read_met_help(ncid, "v", "V", met, met->v, 1.0);
01435   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01436   read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
01437   read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));
01438
01439   /* Meteo data on pressure levels... */
01440   if (ctl->met_np <= 0) {
01441
01442     /* Read pressure levels from file... */
01443     NC(nc_inq_varid(ncid, levname, &varid));
01444     NC(nc_get_var_double(ncid, varid, met->p));
01445     for (ip = 0; ip < met->np; ip++)
01446       met->p[ip] /= 100.;
01447
01448     /* Extrapolate data for lower boundary... */
01449     read_met_extrapolate(met);
01450   }
01451
01452   /* Meteo data on model levels... */
01453   else {
01454
01455     /* Read pressure data from file... */
01456     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01457
01458     /* Interpolate from model levels to pressure levels... */
01459     read_met_ml2pl(ctl, met, met->t);
01460     read_met_ml2pl(ctl, met, met->u);
01461     read_met_ml2pl(ctl, met, met->v);
01462     read_met_ml2pl(ctl, met, met->w);
01463     read_met_ml2pl(ctl, met, met->h2o);
01464     read_met_ml2pl(ctl, met, met->o3);
01465
01466     /* Set pressure levels... */
01467     met->np = ctl->met_np;
01468     for (ip = 0; ip < met->np; ip++)
01469       met->p[ip] = ctl->met_p[ip];
```

```
01470   }
01471
01472   /* Check ordering of pressure levels... */
01473   for (ip = 1; ip < met->np; ip++)
01474     if (met->p[ip - 1] < met->p[ip])
01475       ERRMSG("Pressure levels must be descending!");
01476
01477   /* Read surface pressure... */
01478   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01479       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01480     NC(nc_get_var_float(ncid, varid, help));
01481     for (iy = 0; iy < met->ny; iy++)
01482       for (ix = 0; ix < met->nx; ix++)
01483         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01484   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01485              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01486     NC(nc_get_var_float(ncid, varid, help));
01487     for (iy = 0; iy < met->ny; iy++)
01488       for (ix = 0; ix < met->nx; ix++)
01489         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01490   } else
01491     for (ix = 0; ix < met->nx; ix++)
01492       for (iy = 0; iy < met->ny; iy++)
01493         met->ps[ix][iy] = met->p[0];
01494
01495   /* Create periodic boundary conditions... */
01496   read_met_periodic(met);
01497
01498   /* Calculate geopotential heights... */
01499   read_met_geopot(ctl, met);
01500
01501   /* Calculate potential vorticity... */
01502   read_met_pv(met);
01503
01504   /* Calculate tropopause pressure... */
01505   read_met_tropo(ctl, met);
01506
01507   /* Downsampling... */
01508   read_met_sample(ctl, met);
01509
01510   /* Close file... */
01511   NC(nc_close(ncid));
01512
01513   /* Return success... */
01514   return 1;
01515 }
```

Here is the call graph for this function:



5.19.2.20 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 1519 of file libtrac.c.

```
01520               {
01521
01522   int ip, ip0, ix, iy;
01523
01524   /* Loop over columns... */
01525 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01526   for (ix = 0; ix < met->nx; ix++)
01527     for (iy = 0; iy < met->ny; iy++) {
01528
01529       /* Find lowest valid data point... */
01530       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01531         if (!gsl_finite(met->t[ix][iy][ip0])
01532             || !gsl_finite(met->u[ix][iy][ip0])
01533             || !gsl_finite(met->v[ix][iy][ip0])
01534             || !gsl_finite(met->w[ix][iy][ip0]))
01535           break;
01536
01537       /* Extrapolate... */
01538       for (ip = ip0; ip >= 0; ip--) {
01539         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01540         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01541         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01542         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01543         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01544         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01545       }
01546     }
01547 }
```

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

Calculate geopotential heights.

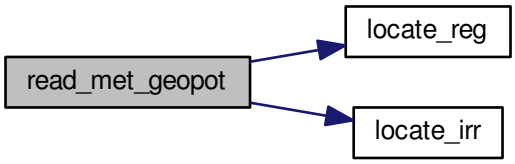
Definition at line 1551 of file libtrac.c.

```
01553               {
01554
01555   static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01556
01557   static int init, topo_nx = -1, topo_ny;
01558
01559   FILE *in;
01560
01561   char line[LEN];
01562
01563   double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01564
01565   float help[EX][EY];
01566
01567   int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01568
01569   /* Initialize geopotential heights... */
01570 #pragma omp parallel for default(shared) private(ix,iy,ip)
01571   for (ix = 0; ix < met->nx; ix++)
01572     for (iy = 0; iy < met->ny; iy++)
01573       for (ip = 0; ip < met->np; ip++)
01574         met->z[ix][iy][ip] = GSL_NAN;
01575
01576   /* Check filename... */
01577   if (ctl->met_geopot[0] == '-')
01578     return;
01579
01580   /* Read surface geopotential... */
01581   if (!init) {
01582     init = 1;
01583
01584     /* Write info... */
01585     printf("Read surface geopotential: %s\n", ctl->met_geopot);
01586
01587     /* Open file... */
```

```
01588     if (!(in = fopen(ctl->met_geopot, "r")))
01589       ERRMSG("Cannot open file!");
01590
01591     /* Read data... */
01592     while (fgets(line, LEN, in))
01593       if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01594         if (rlon != rlon_old) {
01595           if ((++topo_nx) >= EX)
01596             ERRMSG("Too many longitudes!");
01597           topo_ny = 0;
01598         }
01599         rlon_old = rlon;
01600         topo_lon[topo_nx] = rlon;
01601         topo_lat[topo_ny] = rlat;
01602         topo_z[topo_nx][topo_ny] = rz;
01603         if ((++topo_ny) >= EY)
01604           ERRMSG("Too many latitudes!");
01605       }
01606     if ((++topo_nx) >= EX)
01607       ERRMSG("Too many longitudes!");
01608
01609     /* Close file... */
01610     fclose(in);
01611
01612     /* Check grid spacing... */
01613     if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01614         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01615       printf("Warning: Grid spacing does not match!\n");
01616   }
01617
01618   /* Apply hydrostatic equation to calculate geopotential heights... */
01619 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01620   for (ix = 0; ix < met->nx; ix++)
01621     for (iy = 0; iy < met->ny; iy++) {
01622
01623       /* Get surface height... */
01624       lon = met->lon[ix];
01625       if (lon < topo_lon[0])
01626         lon += 360;
01627       else if (lon > topo_lon[topo_nx - 1])
01628         lon -= 360;
01629       lat = met->lat[iy];
01630       tx = locate_reg(topo_lon, topo_nx, lon);
01631       ty = locate_reg(topo_lat, topo_ny, lat);
01632       z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01633                topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01634       z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01635                topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01636       z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01637
01638       /* Find surface pressure level... */
01639       ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01640
01641       /* Get surface temperature... */
01642       ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
01643                met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
01644
01645       /* Upper part of profile... */
01646       met->z[ix][iy][ip0 + 1]
01647         = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01648                    * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01649       for (ip = ip0 + 2; ip < met->np; ip++)
01650         met->z[ix][iy][ip]
01651           = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
01652                      * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01653                      * log(met->p[ip - 1] / met->p[ip]));
01654     }
01655
01656   /* Smooth fields... */
01657   for (ip = 0; ip < met->np; ip++) {
01658
01659     /* Median filter... */
01660 #pragma omp parallel for default(shared) private(ix,iy,n,ix2,ix3,iy2,data)
01661     for (ix = 0; ix < met->nx; ix++)
01662       for (iy = 0; iy < met->ny; iy++) {
01663         n = 0;
01664         for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
01665           ix3 = ix2;
01666           if (ix3 < 0)
01667             ix3 += met->nx;
01668           if (ix3 >= met->nx)
01669             ix3 -= met->nx;
01670           for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01671                iy2++)
01672             if (gsl_finite(met->z[ix3][iy2][ip])) {
01673               data[n] = met->z[ix3][iy2][ip];
01674               n++;
```

```
01675             }
01676         }
01677         if (n > 0) {
01678           gsl_sort(data, 1, (size_t) n);
01679           help[ix][iy] = (float)
01680             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01681         } else
01682           help[ix][iy] = GSL_NAN;
01683       }
01684
01685     /* Copy data... */
01686 #pragma omp parallel for default(shared) private(ix,iy)
01687     for (ix = 0; ix < met->nx; ix++)
01688       for (iy = 0; iy < met->ny; iy++)
01689         met->z[ix][iy][ip] = help[ix][iy];
01690   }
01691 }
```

Here is the call graph for this function:



5.19.2.22 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 1695 of file libtrac.c.

```
01701             {
01702
01703   float *help;
01704
01705   int ip, ix, iy, varid;
01706
01707   /* Check if variable exists... */
01708   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01709     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01710       return;
01711
01712   /* Allocate... */
01713   ALLOC(help, float, met->nx * met->ny * met->np);
01714
01715   /* Read data... */
01716   NC(nc_get_var_float(ncid, varid, help));
01717
01718   /* Copy and check data... */
01719 #pragma omp parallel for default(shared) private(ix,iy,ip)
01720   for (ix = 0; ix < met->nx; ix++)
01721     for (iy = 0; iy < met->ny; iy++)
01722       for (ip = 0; ip < met->np; ip++) {
01723         dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01724         if (fabsf(dest[ix][iy][ip]) < 1e14f)
01725           dest[ix][iy][ip] *= scl;
01726         else
01727           dest[ix][iy][ip] = GSL_NAN;
01728       }
01729
01730   /* Free... */
01731   free(help);
01732 }
```

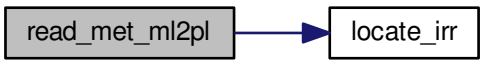
5.19.2.23 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 1736 of file libtrac.c.

```
01739                           {
01740
01741   double aux[EP], p[EP], pt;
01742
01743   int ip, ip2, ix, iy;
01744
01745   /* Loop over columns... */
01746 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01747   for (ix = 0; ix < met->nx; ix++)
01748     for (iy = 0; iy < met->ny; iy++) {
01749
01750       /* Copy pressure profile... */
01751       for (ip = 0; ip < met->np; ip++)
01752         p[ip] = met->pl[ix][iy][ip];
01753
01754       /* Interpolate... */
01755       for (ip = 0; ip < ctl->met_np; ip++) {
01756         pt = ctl->met_p[ip];
01757         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01758           pt = p[0];
01759         else if ((pt > p[met->np - 1] && p[1] > p[0])
01760                  || (pt < p[met->np - 1] && p[1] < p[0]))
01761           pt = p[met->np - 1];
01762         ip2 = locate_irr(p, met->np, pt);
01763         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01764                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01765       }
01766
01767       /* Copy data... */
01768       for (ip = 0; ip < ctl->met_np; ip++)
01769         var[ix][iy][ip] = (float) aux[ip];
01770     }
01771 }
```

Here is the call graph for this function:



5.19.2.24 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 1775 of file libtrac.c.

```
01776                 {
01777
01778   int ip, iy;
01779
01780   /* Check longitudes... */
01781   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01782              + met->lon[1] - met->lon[0] - 360) < 0.01))
01783     return;
01784
01785   /* Increase longitude counter... */
```

```
01786   if ((++met->nx) > EX)
01787     ERRMSG("Cannot create periodic boundary conditions!");
01788
01789   /* Set longitude... */
01790   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01791
01792   /* Loop over latitudes and pressure levels... */
01793 #pragma omp parallel for default(shared) private(iy,ip)
01794   for (iy = 0; iy < met->ny; iy++) {
01795     met->ps[met->nx - 1][iy] = met->ps[0][iy];
01796     met->pt[met->nx - 1][iy] = met->pt[0][iy];
01797     for (ip = 0; ip < met->np; ip++) {
01798       met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01799       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01800       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01801       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01802       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01803       met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01804       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01805       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01806     }
01807   }
01808 }
```

5.19.2.25 void read_met_pv (met_t * *met*)

Calculate potential vorticity.

Definition at line 1812 of file libtrac.c.

```
01813               {
01814
01815   double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01816     dtdp, dudp, dvdp, latr, vort, pows[EP];
01817
01818   int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01819
01820   /* Set powers... */
01821   for (ip = 0; ip < met->np; ip++)
01822     pows[ip] = pow(1000. / met->p[ip], 0.286);
01823
01824   /* Loop over grid points... */
01825 #pragma omp parallel for default(shared)
       private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
01826   for (ix = 0; ix < met->nx; ix++) {
01827
01828     /* Set indices... */
01829     ix0 = GSL_MAX(ix - 1, 0);
01830     ix1 = GSL_MIN(ix + 1, met->nx - 1);
01831
01832     /* Loop over grid points... */
01833     for (iy = 0; iy < met->ny; iy++) {
01834
01835       /* Set indices... */
01836       iy0 = GSL_MAX(iy - 1, 0);
01837       iy1 = GSL_MIN(iy + 1, met->ny - 1);
01838
01839       /* Set auxiliary variables... */
01840       latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01841       dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
01842       dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01843       c0 = cos(met->lat[iy0] / 180. * M_PI);
01844       c1 = cos(met->lat[iy1] / 180. * M_PI);
01845       cr = cos(latr / 180. * M_PI);
01846       vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01847
01848       /* Loop over grid points... */
01849       for (ip = 0; ip < met->np; ip++) {
01850
01851         /* Get gradients in longitude... */
01852         dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
01853         dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01854
01855         /* Get gradients in latitude... */
01856         dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01857         dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01858
01859         /* Set indices... */
01860         ip0 = GSL_MAX(ip - 1, 0);
```

```
01861         ip1 = GSL_MIN(ip + 1, met->np - 1);
01862
01863         /* Get gradients in pressure... */
01864         dp0 = 100. * (met->p[ip] - met->p[ip0]);
01865         dp1 = 100. * (met->p[ip1] - met->p[ip]);
01866         if (ip != ip0 && ip != ip1) {
01867           denom = dp0 * dp1 * (dp0 + dp1);
01868           dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01869                   - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01870                   + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01871             / denom;
01872           dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01873                   - dp1 * dp1 * met->u[ix][iy][ip0]
01874                   + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01875             / denom;
01876           dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
01877                   - dp1 * dp1 * met->v[ix][iy][ip0]
01878                   + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01879             / denom;
01880         } else {
01881           denom = dp0 + dp1;
01882           dtdp =
01883             (met->t[ix][iy][ip1] * pows[ip1] -
01884              met->t[ix][iy][ip0] * pows[ip0]) / denom;
01885           dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01886           dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01887         }
01888
01889         /* Calculate PV... */
01890         met->pv[ix][iy][ip] = (float)
01891           (1e6 * G0 *
01892            (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01893       }
01894     }
01895   }
01896 }
```

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

Downsampling of meteorological data.

Definition at line 1900 of file libtrac.c.

```
01902               {
01903
01904   met_t *help;
01905
01906   float w, wsum;
01907
01908   int ip, ip2, ix, ix2, ix3, iy, iy2;
01909
01910   /* Check parameters... */
01911   if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
01912       && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01913     return;
01914
01915   /* Allocate... */
01916   ALLOC(help, met_t, 1);
01917
01918   /* Copy data... */
01919   help->nx = met->nx;
01920   help->ny = met->ny;
01921   help->np = met->np;
01922   memcpy(help->lon, met->lon, sizeof(met->lon));
01923   memcpy(help->lat, met->lat, sizeof(met->lat));
01924   memcpy(help->p, met->p, sizeof(met->p));
01925
01926   /* Smoothing... */
01927   for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01928     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01929       for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01930         help->ps[ix][iy] = 0;
01931         help->pt[ix][iy] = 0;
01932         help->z[ix][iy][ip] = 0;
01933         help->t[ix][iy][ip] = 0;
01934         help->u[ix][iy][ip] = 0;
01935         help->v[ix][iy][ip] = 0;
01936         help->w[ix][iy][ip] = 0;
01937         help->pv[ix][iy][ip] = 0;
01938         help->h2o[ix][iy][ip] = 0;
```

```
01939         help->o3[ix][iy][ip] = 0;
01940         wsum = 0;
01941         for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
01942           ix3 = ix2;
01943           if (ix3 < 0)
01944             ix3 += met->nx;
01945           else if (ix3 >= met->nx)
01946             ix3 -= met->nx;
01947
01948           for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01949                iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
01950             for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
01951                  ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
01952               w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
01953                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
01954                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01955               help->ps[ix][iy] += w * met->ps[ix3][iy2];
01956               help->pt[ix][iy] += w * met->pt[ix3][iy2];
01957               help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
01958               help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
01959               help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
01960               help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
01961               help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
01962               help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
01963               help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
01964               help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
01965               wsum += w;
01966             }
01967         }
01968         help->ps[ix][iy] /= wsum;
01969         help->pt[ix][iy] /= wsum;
01970         help->t[ix][iy][ip] /= wsum;
01971         help->z[ix][iy][ip] /= wsum;
01972         help->u[ix][iy][ip] /= wsum;
01973         help->v[ix][iy][ip] /= wsum;
01974         help->w[ix][iy][ip] /= wsum;
01975         help->pv[ix][iy][ip] /= wsum;
01976         help->h2o[ix][iy][ip] /= wsum;
01977         help->o3[ix][iy][ip] /= wsum;
01978       }
01979     }
01980   }
01981
01982   /* Downsampling... */
01983   met->nx = 0;
01984   for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01985     met->lon[met->nx] = help->lon[ix];
01986     met->ny = 0;
01987     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01988       met->lat[met->ny] = help->lat[iy];
01989       met->ps[met->nx][met->ny] = help->ps[ix][iy];
01990       met->pt[met->nx][met->ny] = help->pt[ix][iy];
01991       met->np = 0;
01992       for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01993         met->p[met->np] = help->p[ip];
01994         met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01995         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01996         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01997         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01998         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01999         met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
02000         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
02001         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
02002         met->np++;
02003       }
02004       met->ny++;
02005     }
02006     met->nx++;
02007   }
02008
02009   /* Free... */
02010   free(help);
02011 }
```

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

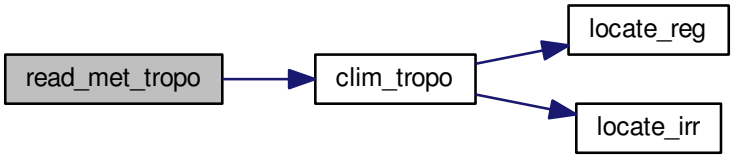
Calculate tropopause pressure.

Definition at line 2015 of file libtrac.c.

```
02017               {
02018
02019     gsl_interp_accel *acc;
02020
02021     gsl_spline *spline;
02022
02023     double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
02024       z[400], z2[400];
02025
02026     int found, ix, iy, iz, iz2;
02027
02028     /* Allocate... */
02029     acc = gsl_interp_accel_alloc();
02030     spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
02031
02032     /* Get altitude and pressure profiles... */
02033     for (iz = 0; iz < met->np; iz++)
02034       z[iz] = Z(met->p[iz]);
02035     for (iz = 0; iz <= 170; iz++) {
02036       z2[iz] = 4.5 + 0.1 * iz;
02037       p2[iz] = P(z2[iz]);
02038     }
02039
02040     /* Do not calculate tropopause... */
02041     if (ctl->met_tropo == 0)
02042       for (ix = 0; ix < met->nx; ix++)
02043         for (iy = 0; iy < met->ny; iy++)
02044           met->pt[ix][iy] = GSL_NAN;
02045
02046     /* Use tropopause climatology... */
02047     else if (ctl->met_tropo == 1)
02048       for (ix = 0; ix < met->nx; ix++)
02049         for (iy = 0; iy < met->ny; iy++)
02050           met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
02051
02052     /* Use cold point... */
02053     else if (ctl->met_tropo == 2) {
02054
02055       /* Loop over grid points... */
02056       for (ix = 0; ix < met->nx; ix++)
02057         for (iy = 0; iy < met->ny; iy++) {
02058
02059           /* Interpolate temperature profile... */
02060           for (iz = 0; iz < met->np; iz++)
02061             t[iz] = met->t[ix][iy][iz];
02062           gsl_spline_init(spline, z, t, (size_t) met->np);
02063           for (iz = 0; iz <= 170; iz++)
02064             t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02065
02066           /* Find minimum... */
02067           iz = (int) gsl_stats_min_index(t2, 1, 171);
02068           if (iz <= 0 || iz >= 170)
02069             met->pt[ix][iy] = GSL_NAN;
02070           else
02071             met->pt[ix][iy] = p2[iz];
02072         }
02073     }
02074
02075     /* Use WMO definition... */
02076     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02077
02078       /* Loop over grid points... */
02079       for (ix = 0; ix < met->nx; ix++)
02080         for (iy = 0; iy < met->ny; iy++) {
02081
02082           /* Interpolate temperature profile... */
02083           for (iz = 0; iz < met->np; iz++)
02084             t[iz] = met->t[ix][iy][iz];
02085           gsl_spline_init(spline, z, t, (size_t) met->np);
02086           for (iz = 0; iz <= 160; iz++)
02087             t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02088
02089           /* Find 1st tropopause... */
02090           met->pt[ix][iy] = GSL_NAN;
02091           for (iz = 0; iz <= 140; iz++) {
02092             found = 1;
02093             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02094               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02095                   / log(p2[iz2] / p2[iz]) > 2.0) {
02096                 found = 0;
02097                 break;
02098               }
02099             if (found) {
02100               if (iz > 0 && iz < 140)
02101                 met->pt[ix][iy] = p2[iz];
02102               break;
02103             }
```

```
02104         }
02105
02106         /* Find 2nd tropopause... */
02107         if (ctl->met_tropo == 4) {
02108           met->pt[ix][iy] = GSL_NAN;
02109           for (; iz <= 140; iz++) {
02110             found = 1;
02111             for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
02112               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02113                   / log(p2[iz2] / p2[iz]) < 3.0) {
02114                 found = 0;
02115                 break;
02116               }
02117             if (found)
02118               break;
02119           }
02120           for (; iz <= 140; iz++) {
02121             found = 1;
02122             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02123               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02124                   / log(p2[iz2] / p2[iz]) > 2.0) {
02125                 found = 0;
02126                 break;
02127               }
02128             if (found) {
02129               if (iz > 0 && iz < 140)
02130                 met->pt[ix][iy] = p2[iz];
02131               break;
02132             }
02133           }
02134         }
02135       }
02136   }
02137
02138   /* Use dynamical tropopause... */
02139   else if (ctl->met_tropo == 5) {
02140
02141     /* Loop over grid points... */
02142     for (ix = 0; ix < met->nx; ix++)
02143       for (iy = 0; iy < met->ny; iy++) {
02144
02145         /* Interpolate potential vorticity profile... */
02146         for (iz = 0; iz < met->np; iz++)
02147           pv[iz] = met->pv[ix][iy][iz];
02148         gsl_spline_init(spline, z, pv, (size_t) met->np);
02149         for (iz = 0; iz <= 160; iz++)
02150           pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02151
02152         /* Interpolate potential temperature profile... */
02153         for (iz = 0; iz < met->np; iz++)
02154           th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02155         gsl_spline_init(spline, z, th, (size_t) met->np);
02156         for (iz = 0; iz <= 160; iz++)
02157           th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02158
02159         /* Find dynamical tropopause 3.5 PVU + 380 K */
02160         met->pt[ix][iy] = GSL_NAN;
02161         for (iz = 0; iz <= 160; iz++)
02162           if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02163             if (iz > 0 && iz < 160)
02164               met->pt[ix][iy] = p2[iz];
02165             break;
02166           }
02167       }
02168   }
02169
02170   else
02171     ERRMSG("Cannot calculate tropopause!");
02172
02173   /* Free... */
02174   gsl_spline_free(spline);
02175   gsl_interp_accel_free(acc);
02176 }
```

Here is the call graph for this function:



5.19.2.28 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 2180 of file libtrac.c.

```
02187               {
02188
02189   FILE *in = NULL;
02190
02191   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02192     msg[2 * LEN], rvarname[LEN], rval[LEN];
02193
02194   int contain = 0, i;
02195
02196   /* Open file... */
02197   if (filename[strlen(filename) - 1] != '-')
02198     if (!(in = fopen(filename, "r")))
02199       ERRMSG("Cannot open file!");
02200
02201   /* Set full variable name... */
02202   if (arridx >= 0) {
02203     sprintf(fullname1, "%s[%d]", varname, arridx);
02204     sprintf(fullname2, "%s[*]", varname);
02205   } else {
02206     sprintf(fullname1, "%s", varname);
02207     sprintf(fullname2, "%s", varname);
02208   }
02209
02210   /* Read data... */
02211   if (in != NULL)
02212     while (fgets(line, LEN, in))
02213       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02214         if (strcasecmp(rvarname, fullname1) == 0 ||
02215             strcasecmp(rvarname, fullname2) == 0) {
02216           contain = 1;
02217           break;
02218         }
02219   for (i = 1; i < argc - 1; i++)
02220     if (strcasecmp(argv[i], fullname1) == 0 ||
02221         strcasecmp(argv[i], fullname2) == 0) {
02222       sprintf(rval, "%s", argv[i + 1]);
02223       contain = 1;
02224       break;
02225     }
02226
02227   /* Close file... */
02228   if (in != NULL)
02229     fclose(in);
02230
02231   /* Check for missing variables... */
02232   if (!contain) {
02233     if (strlen(defvalue) > 0)
02234       sprintf(rval, "%s", defvalue);
02235     else {
02236       sprintf(msg, "Missing variable %s!\n", fullname1);
02237       ERRMSG(msg);
```

```
02238     }
02239   }
02240
02241   /* Write info... */
02242   printf("%s = %s\n", fullname1, rval);
02243
02244   /* Return values... */
02245   if (value != NULL)
02246     sprintf(value, "%s", rval);
02247   return atof(rval);
02248 }
```

5.19.2.29 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 2252 of file libtrac.c.

```
02260                 {
02261
02262   struct tm t0, t1;
02263
02264   t0.tm_year = 100;
02265   t0.tm_mon = 0;
02266   t0.tm_mday = 1;
02267   t0.tm_hour = 0;
02268   t0.tm_min = 0;
02269   t0.tm_sec = 0;
02270
02271   t1.tm_year = year - 1900;
02272   t1.tm_mon = mon - 1;
02273   t1.tm_mday = day;
02274   t1.tm_hour = hour;
02275   t1.tm_min = min;
02276   t1.tm_sec = sec;
02277
02278   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02279 }
```

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

Measure wall-clock time.

Definition at line 2283 of file libtrac.c.

```
02286             {
02287
02288   static double starttime[NTIMER], runtime[NTIMER];
02289
02290   /* Check id... */
02291   if (id < 0 || id >= NTIMER)
02292     ERRMSG("Too many timers!");
02293
02294   /* Start timer... */
02295   if (mode == 1) {
02296     if (starttime[id] <= 0)
02297       starttime[id] = omp_get_wtime();
02298     else
02299       ERRMSG("Timer already started!");
02300   }
02301
02302   /* Stop timer... */
02303   else if (mode == 2) {
02304     if (starttime[id] > 0) {
02305       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02306       starttime[id] = -1;
02307     }
02308   }
02309
02310   /* Print timer... */
02311   else if (mode == 3) {
02312     printf("%s = %.3f s\n", name, runtime[id]);
02313     runtime[id] = 0;
02314   }
02315 }
```

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

Write atmospheric data.

Definition at line 2319 of file libtrac.c.

```
02323           {
02324
02325   FILE *in, *out;
02326
02327   char line[LEN];
02328
02329   double r, t0, t1;
02330
02331   int ip, iq, year, mon, day, hour, min, sec;
02332
02333   /* Set time interval for output... */
02334   t0 = t - 0.5 * ctl->dt_mod;
02335   t1 = t + 0.5 * ctl->dt_mod;
02336
02337   /* Write info... */
02338   printf("Write atmospheric data: %s\n", filename);
02339
02340   /* Write ASCII data... */
02341   if (ctl->atm_type == 0) {
02342
02343     /* Check if gnuplot output is requested... */
02344     if (ctl->atm_gpfile[0] != '-') {
02345
02346       /* Create gnuplot pipe... */
02347       if (!(out = popen("gnuplot", "w")))
02348         ERRMSG("Cannot create pipe to gnuplot!");
02349
02350       /* Set plot filename... */
02351       fprintf(out, "set out \"%s.png\"\n", filename);
02352
02353       /* Set time string... */
02354       jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02355       fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02356               year, mon, day, hour, min);
02357
02358       /* Dump gnuplot file to pipe... */
02359       if (!(in = fopen(ctl->atm_gpfile, "r")))
02360         ERRMSG("Cannot open file!");
02361       while (fgets(line, LEN, in))
02362         fprintf(out, "%s", line);
02363       fclose(in);
02364     }
02365
02366     else {
02367
02368       /* Create file... */
02369       if (!(out = fopen(filename, "w")))
02370         ERRMSG("Cannot create file!");
02371     }
02372
02373     /* Write header... */
02374     fprintf(out,
02375             "# $1 = time [s]\n"
02376             "# $2 = altitude [km]\n"
02377             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02378     for (iq = 0; iq < ctl->nq; iq++)
02379       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02380               ctl->qnt_unit[iq]);
02381     fprintf(out, "\n");
02382
02383     /* Write data... */
02384     for (ip = 0; ip < atm->np; ip++) {
02385
02386       /* Check time... */
02387       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02388         continue;
02389
02390       /* Write output... */
02391       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02392               atm->lon[ip], atm->lat[ip]);
02393       for (iq = 0; iq < ctl->nq; iq++) {
02394         fprintf(out, " ");
02395         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02396       }
02397       fprintf(out, "\n");
02398     }
02399
```

```
02400     /* Close file... */
02401     fclose(out);
02402   }
02403
02404   /* Write binary data... */
02405   else if (ctl->atm_type == 1) {
02406
02407     /* Create file... */
02408     if (!(out = fopen(filename, "w")))
02409       ERRMSG("Cannot create file!");
02410
02411     /* Write data... */
02412     FWRITE(&atm->np, int,
02413            1,
02414            out);
02415     FWRITE(atm->time, double,
02416              (size_t) atm->np,
02417            out);
02418     FWRITE(atm->p, double,
02419              (size_t) atm->np,
02420            out);
02421     FWRITE(atm->lon, double,
02422              (size_t) atm->np,
02423            out);
02424     FWRITE(atm->lat, double,
02425              (size_t) atm->np,
02426            out);
02427     for (iq = 0; iq < ctl->nq; iq++)
02428       FWRITE(atm->q[iq], double,
02429                (size_t) atm->np,
02430              out);
02431
02432     /* Close file... */
02433     fclose(out);
02434   }
02435
02436   /* Error... */
02437   else
02438     ERRMSG("Atmospheric data type not supported!");
02439 }
```

Here is the call graph for this function:

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

Write CSI data.

Definition at line 2443 of file libtrac.c.

```
02447             {
02448
02449   static FILE *in, *out;
02450
02451   static char line[LEN];
02452
02453   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02454     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02455
02456   static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02457
02458   /* Init... */
02459   if (t == ctl->t_start) {
```

```
02460
02461     /* Check quantity index for mass... */
02462     if (ctl->qnt_m < 0)
02463       ERRMSG("Need quantity mass!");
02464
02465     /* Open observation data file... */
02466     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02467     if (!(in = fopen(ctl->csi_obsfile, "r")))
02468       ERRMSG("Cannot open file!");
02469
02470     /* Create new file... */
02471     printf("Write CSI data: %s\n", filename);
02472     if (!(out = fopen(filename, "w")))
02473       ERRMSG("Cannot create file!");
02474
02475     /* Write header... */
02476     fprintf(out,
02477             "# $1 = time [s]\n"
02478             "# $2 = number of hits (cx)\n"
02479             "# $3 = number of misses (cy)\n"
02480             "# $4 = number of false alarms (cz)\n"
02481             "# $5 = number of observations (cx + cy)\n"
02482             "# $6 = number of forecasts (cx + cz)\n"
02483             "# $7 = bias (forecasts/observations) [%%]\n"
02484             "# $8 = probability of detection (POD) [%%]\n"
02485             "# $9 = false alarm rate (FAR) [%%]\n"
02486             "# $10 = critical success index (CSI) [%%]\n\n");
02487   }
02488
02489   /* Set time interval... */
02490   t0 = t - 0.5 * ctl->dt_mod;
02491   t1 = t + 0.5 * ctl->dt_mod;
02492
02493   /* Initialize grid cells... */
02494 #pragma omp parallel for default(shared) private(ix,iy,iz)
02495   for (ix = 0; ix < ctl->csi_nx; ix++)
02496     for (iy = 0; iy < ctl->csi_ny; iy++)
02497       for (iz = 0; iz < ctl->csi_nz; iz++)
02498         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02499
02500   /* Read observation data... */
02501   while (fgets(line, LEN, in)) {
02502
02503     /* Read data... */
02504     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02505         5)
02506       continue;
02507
02508     /* Check time... */
02509     if (rt < t0)
02510       continue;
02511     if (rt > t1)
02512       break;
02513
02514     /* Calculate indices... */
02515     ix = (int) ((rlon - ctl->csi_lon0)
02516                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02517     iy = (int) ((rlat - ctl->csi_lat0)
02518                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02519     iz = (int) ((rz - ctl->csi_z0)
02520                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02521
02522     /* Check indices... */
02523     if (ix < 0 || ix >= ctl->csi_nx ||
02524         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02525       continue;
02526
02527     /* Get mean observation index... */
02528     obsmean[ix][iy][iz] += robs;
02529     obscount[ix][iy][iz]++;
02530   }
02531
02532   /* Analyze model data... */
02533 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02534   for (ip = 0; ip < atm->np; ip++) {
02535
02536     /* Check time... */
02537     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02538       continue;
02539
02540     /* Get indices... */
02541     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02542                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02543     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02544                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02545     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02546                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
```

```
02547
02548     /* Check indices... */
02549     if (ix < 0 || ix >= ctl->csi_nx ||
02550         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02551       continue;
02552
02553     /* Get total mass in grid cell... */
02554     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02555   }
02556
02557   /* Analyze all grid cells... */
02558 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02559   for (ix = 0; ix < ctl->csi_nx; ix++)
02560     for (iy = 0; iy < ctl->csi_ny; iy++)
02561       for (iz = 0; iz < ctl->csi_nz; iz++) {
02562
02563         /* Calculate mean observation index... */
02564         if (obscount[ix][iy][iz] > 0)
02565           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02566
02567         /* Calculate column density... */
02568         if (modmean[ix][iy][iz] > 0) {
02569           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02570           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02571           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02572           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02573             * cos(lat * M_PI / 180.);
02574           modmean[ix][iy][iz] /= (1e6 * area);
02575         }
02576
02577         /* Calculate CSI... */
02578         if (obscount[ix][iy][iz] > 0) {
02579           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02580               modmean[ix][iy][iz] >= ctl->csi_modmin)
02581             cx++;
02582           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02583                    modmean[ix][iy][iz] < ctl->csi_modmin)
02584             cy++;
02585           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02586                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02587             cz++;
02588         }
02589       }
02590
02591   /* Write output... */
02592   if (fmod(t, ctl->csi_dt_out) == 0) {
02593
02594     /* Write... */
02595     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02596             t, cx, cy, cz, cx + cy, cx + cz,
02597             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02598             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02599             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02600             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02601
02602     /* Set counters to zero... */
02603     cx = cy = cz = 0;
02604   }
02605
02606   /* Close file... */
02607   if (t == ctl->t_stop)
02608     fclose(out);
02609 }
```

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

Write ensemble data.

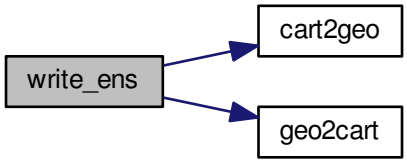
Definition at line 2613 of file libtrac.c.

```
02617             {
02618
02619   static FILE *out;
02620
02621   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02622     t0, t1, x[NENS][3], xm[3];
02623
02624   static int ip, iq;
02625
02626   static size_t i, n;
```

```
02627
02628   /* Init... */
02629   if (t == ctl->t_start) {
02630
02631     /* Check quantities... */
02632     if (ctl->qnt_ens < 0)
02633       ERRMSG("Missing ensemble IDs!");
02634
02635     /* Create new file... */
02636     printf("Write ensemble data: %s\n", filename);
02637     if (!(out = fopen(filename, "w")))
02638       ERRMSG("Cannot create file!");
02639
02640     /* Write header... */
02641     fprintf(out,
02642             "# $1 = time [s]\n"
02643             "# $2 = altitude [km]\n"
02644             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02645     for (iq = 0; iq < ctl->nq; iq++)
02646       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02647               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02648     for (iq = 0; iq < ctl->nq; iq++)
02649       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02650               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02651     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02652   }
02653
02654   /* Set time interval... */
02655   t0 = t - 0.5 * ctl->dt_mod;
02656   t1 = t + 0.5 * ctl->dt_mod;
02657
02658   /* Init... */
02659   ens = GSL_NAN;
02660   n = 0;
02661
02662   /* Loop over air parcels... */
02663   for (ip = 0; ip < atm->np; ip++) {
02664
02665     /* Check time... */
02666     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02667       continue;
02668
02669     /* Check ensemble id... */
02670     if (atm->q[ctl->qnt_ens][ip] != ens) {
02671
02672       /* Write results... */
02673       if (n > 0) {
02674
02675         /* Get mean position... */
02676         xm[0] = xm[1] = xm[2] = 0;
02677         for (i = 0; i < n; i++) {
02678           xm[0] += x[i][0] / (double) n;
02679           xm[1] += x[i][1] / (double) n;
02680           xm[2] += x[i][2] / (double) n;
02681         }
02682         cart2geo(xm, &dummy, &lon, &lat);
02683         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02684                 lat);
02685
02686         /* Get quantity statistics... */
02687         for (iq = 0; iq < ctl->nq; iq++) {
02688           fprintf(out, " ");
02689           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02690         }
02691         for (iq = 0; iq < ctl->nq; iq++) {
02692           fprintf(out, " ");
02693           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02694         }
02695         fprintf(out, " %lu\n", n);
02696       }
02697
02698       /* Init new ensemble... */
02699       ens = atm->q[ctl->qnt_ens][ip];
02700       n = 0;
02701     }
02702
02703     /* Save data... */
02704     p[n] = atm->p[ip];
02705     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02706     for (iq = 0; iq < ctl->nq; iq++)
02707       q[iq][n] = atm->q[iq][ip];
02708     if ((++n) >= NENS)
02709       ERRMSG("Too many data points!");
02710   }
02711
02712   /* Write results... */
02713   if (n > 0) {
```

```
02714
02715     /* Get mean position... */
02716     xm[0] = xm[1] = xm[2] = 0;
02717     for (i = 0; i < n; i++) {
02718       xm[0] += x[i][0] / (double) n;
02719       xm[1] += x[i][1] / (double) n;
02720       xm[2] += x[i][2] / (double) n;
02721     }
02722     cart2geo(xm, &dummy, &lon, &lat);
02723     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02724
02725     /* Get quantity statistics... */
02726     for (iq = 0; iq < ctl->nq; iq++) {
02727       fprintf(out, " ");
02728       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02729     }
02730     for (iq = 0; iq < ctl->nq; iq++) {
02731       fprintf(out, " ");
02732       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02733     }
02734     fprintf(out, " %lu\n", n);
02735   }
02736
02737   /* Close file... */
02738   if (t == ctl->t_stop)
02739     fclose(out);
02740 }
```

Here is the call graph for this function:



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

Write gridded data.

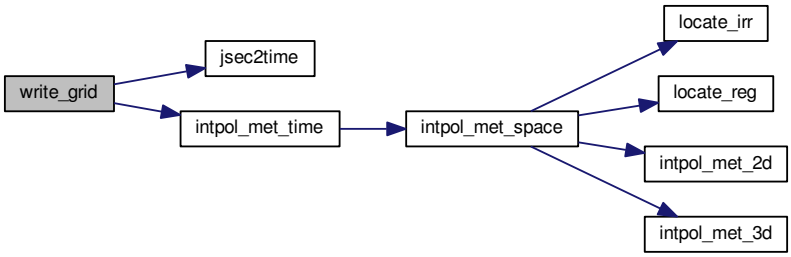
Definition at line 2744 of file libtrac.c.

```
02750             {
02751
02752   FILE *in, *out;
02753
02754   char line[LEN];
02755
02756   static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02757     area, rho_air, press, temp, cd, vmr, t0, t1, r;
02758
02759   static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
02760
02761   /* Check dimensions... */
02762   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02763     ERRMSG("Grid dimensions too large!");
02764
02765   /* Set time interval for output... */
02766   t0 = t - 0.5 * ctl->dt_mod;
02767   t1 = t + 0.5 * ctl->dt_mod;
02768
02769   /* Set grid box size... */
02770   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02771   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
```

```
02772   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02773
02774   /* Initialize grid... */
02775 #pragma omp parallel for default(shared) private(ix,iy,iz)
02776   for (ix = 0; ix < ctl->grid_nx; ix++)
02777     for (iy = 0; iy < ctl->grid_ny; iy++)
02778       for (iz = 0; iz < ctl->grid_nz; iz++) {
02779         mass[ix][iy][iz] = 0;
02780         np[ix][iy][iz] = 0;
02781       }
02782
02783   /* Average data... */
02784 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02785   for (ip = 0; ip < atm->np; ip++)
02786     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02787
02788       /* Get index... */
02789       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02790       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02791       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02792
02793       /* Check indices... */
02794       if (ix < 0 || ix >= ctl->grid_nx ||
02795           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02796         continue;
02797
02798       /* Add mass... */
02799       if (ctl->qnt_m >= 0)
02800         mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02801       np[ix][iy][iz]++;
02802     }
02803
02804   /* Check if gnuplot output is requested... */
02805   if (ctl->grid_gpfile[0] != '-') {
02806
02807     /* Write info... */
02808     printf("Plot grid data: %s.png\n", filename);
02809
02810     /* Create gnuplot pipe... */
02811     if (!(out = popen("gnuplot", "w")))
02812       ERRMSG("Cannot create pipe to gnuplot!");
02813
02814     /* Set plot filename... */
02815     fprintf(out, "set out \"%s.png\"\n", filename);
02816
02817     /* Set time string... */
02818     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02819     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02820             year, mon, day, hour, min);
02821
02822     /* Dump gnuplot file to pipe... */
02823     if (!(in = fopen(ctl->grid_gpfile, "r")))
02824       ERRMSG("Cannot open file!");
02825     while (fgets(line, LEN, in))
02826       fprintf(out, "%s", line);
02827     fclose(in);
02828   }
02829
02830   else {
02831
02832     /* Write info... */
02833     printf("Write grid data: %s\n", filename);
02834
02835     /* Create file... */
02836     if (!(out = fopen(filename, "w")))
02837       ERRMSG("Cannot create file!");
02838   }
02839
02840   /* Write header... */
02841   fprintf(out,
02842           "# $1 = time [s]\n"
02843           "# $2 = altitude [km]\n"
02844           "# $3 = longitude [deg]\n"
02845           "# $4 = latitude [deg]\n"
02846           "# $5 = surface area [km^2]\n"
02847           "# $6 = layer width [km]\n"
02848           "# $7 = number of particles [1]\n"
02849           "# $8 = column density [kg/m^2]\n"
02850           "# $9 = volume mixing ratio [1]\n\n");
02851
02852   /* Write data... */
02853   for (ix = 0; ix < ctl->grid_nx; ix++) {
02854     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02855       fprintf(out, "\n");
02856     for (iy = 0; iy < ctl->grid_ny; iy++) {
02857       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02858         fprintf(out, "\n");
```

```
02859       for (iz = 0; iz < ctl->grid_nz; iz++)
02860         if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02861
02862           /* Set coordinates... */
02863           z = ctl->grid_z0 + dz * (iz + 0.5);
02864           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02865           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02866
02867           /* Get pressure and temperature... */
02868           press = P(z);
02869           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02870                           NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02871
02872           /* Calculate surface area... */
02873           area = dlat * dlon * SQR(RE * M_PI / 180.)
02874             * cos(lat * M_PI / 180.);
02875
02876           /* Calculate column density... */
02877           cd = mass[ix][iy][iz] / (1e6 * area);
02878
02879           /* Calculate volume mixing ratio... */
02880           rho_air = 100. * press / (RA * temp);
02881           vmr = MA / ctl->molmass * mass[ix][iy][iz]
02882             / (rho_air * 1e6 * area * 1e3 * dz);
02883
02884           /* Write output... */
02885           fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
02886                   t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
02887         }
02888     }
02889   }
02890
02891   /* Close file... */
02892   fclose(out);
02893 }
```

Here is the call graph for this function:



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

Write profile data.

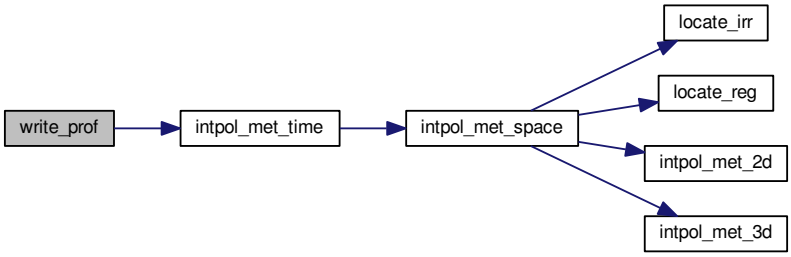
Definition at line 2897 of file libtrac.c.

```
02903             {
02904
02905   static FILE *in, *out;
02906
02907   static char line[LEN];
02908
02909   static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
02910     rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
02911     rho_air, vmr, h2o, o3;
02912
02913   static int obscount[GX][GY], ip, ix, iy, iz, okay;
02914
```

```
02915   /* Init... */
02916   if (t == ctl->t_start) {
02917
02918     /* Check quantity index for mass... */
02919     if (ctl->qnt_m < 0)
02920       ERRMSG("Need quantity mass!");
02921
02922     /* Check dimensions... */
02923     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02924       ERRMSG("Grid dimensions too large!");
02925
02926     /* Open observation data file... */
02927     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02928     if (!(in = fopen(ctl->prof_obsfile, "r")))
02929       ERRMSG("Cannot open file!");
02930
02931     /* Create new output file... */
02932     printf("Write profile data: %s\n", filename);
02933     if (!(out = fopen(filename, "w")))
02934       ERRMSG("Cannot create file!");
02935
02936     /* Write header... */
02937     fprintf(out,
02938             "# $1 = time [s]\n"
02939             "# $2 = altitude [km]\n"
02940             "# $3 = longitude [deg]\n"
02941             "# $4 = latitude [deg]\n"
02942             "# $5 = pressure [hPa]\n"
02943             "# $6 = temperature [K]\n"
02944             "# $7 = volume mixing ratio [1]\n"
02945             "# $8 = H2O volume mixing ratio [1]\n"
02946             "# $9 = O3 volume mixing ratio [1]\n"
02947             "# $10 = observed BT index (mean) [K]\n"
02948             "# $11 = observed BT index (sigma) [K]\n");
02949
02950     /* Set grid box size... */
02951     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
02952     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02953     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02954   }
02955
02956   /* Set time interval... */
02957   t0 = t - 0.5 * ctl->dt_mod;
02958   t1 = t + 0.5 * ctl->dt_mod;
02959
02960   /* Initialize... */
02961 #pragma omp parallel for default(shared) private(ix,iy,iz)
02962   for (ix = 0; ix < ctl->prof_nx; ix++)
02963     for (iy = 0; iy < ctl->prof_ny; iy++) {
02964       obsmean[ix][iy] = 0;
02965       obsmean2[ix][iy] = 0;
02966       obscount[ix][iy] = 0;
02967       for (iz = 0; iz < ctl->prof_nz; iz++)
02968         mass[ix][iy][iz] = 0;
02969     }
02970
02971   /* Read observation data... */
02972   while (fgets(line, LEN, in)) {
02973
02974     /* Read data... */
02975     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02976         5)
02977       continue;
02978
02979     /* Check time... */
02980     if (rt < t0)
02981       continue;
02982     if (rt > t1)
02983       break;
02984
02985     /* Calculate indices... */
02986     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
02987     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02988
02989     /* Check indices... */
02990     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02991       continue;
02992
02993     /* Get mean observation index... */
02994     obsmean[ix][iy] += robs;
02995     obsmean2[ix][iy] += SQR(robs);
02996     obscount[ix][iy]++;
02997   }
02998
02999   /* Analyze model data... */
03000 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03001   for (ip = 0; ip < atm->np; ip++) {
```

```
03002
03003     /* Check time... */
03004     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03005       continue;
03006
03007     /* Get indices... */
03008     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
03009     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
03010     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
03011
03012     /* Check indices... */
03013     if (ix < 0 || ix >= ctl->prof_nx ||
03014         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
03015       continue;
03016
03017     /* Get total mass in grid cell... */
03018     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03019   }
03020
03021   /* Extract profiles... */
03022   for (ix = 0; ix < ctl->prof_nx; ix++)
03023     for (iy = 0; iy < ctl->prof_ny; iy++)
03024       if (obscount[ix][iy] > 0) {
03025
03026         /* Check profile... */
03027         okay = 0;
03028         for (iz = 0; iz < ctl->prof_nz; iz++)
03029           if (mass[ix][iy][iz] > 0) {
03030             okay = 1;
03031             break;
03032           }
03033         if (!okay)
03034           continue;
03035
03036         /* Write output... */
03037         fprintf(out, "\n");
03038
03039         /* Loop over altitudes... */
03040         for (iz = 0; iz < ctl->prof_nz; iz++) {
03041
03042           /* Set coordinates... */
03043           z = ctl->prof_z0 + dz * (iz + 0.5);
03044           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
03045           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
03046
03047           /* Get pressure and temperature... */
03048           press = P(z);
03049           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
03050                           NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);
03051
03052           /* Calculate surface area... */
03053           area = dlat * dlon * SQR(M_PI * RE / 180.)
03054             * cos(lat * M_PI / 180.);
03055
03056           /* Calculate volume mixing ratio... */
03057           rho_air = 100. * press / (RA * temp);
03058           vmr = MA / ctl->molmass * mass[ix][iy][iz]
03059             / (rho_air * area * dz * 1e9);
03060
03061           /* Write output... */
03062           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n",
03063                   t, z, lon, lat, press, temp, vmr, h2o, o3,
03064                   obsmean[ix][iy] / obscount[ix][iy],
03065                   sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03066                        - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03067         }
03068       }
03069
03070   /* Close file... */
03071   if (t == ctl->t_stop)
03072     fclose(out);
03073 }
```

Here is the call graph for this function:



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

Write station data.

Definition at line 3077 of file libtrac.c.

```
03081            {
03082
03083   static FILE *out;
03084
03085   static double rmax2, t0, t1, x0[3], x1[3];
03086
03087   static int ip, iq;
03088
03089   /* Init... */
03090   if (t == ctl->t_start) {
03091
03092     /* Write info... */
03093     printf("Write station data: %s\n", filename);
03094
03095     /* Create new file... */
03096     if (!(out = fopen(filename, "w")))
03097       ERRMSG("Cannot create file!");
03098
03099     /* Write header... */
03100     fprintf(out,
03101             "# $1 = time [s]\n"
03102             "# $2 = altitude [km]\n"
03103             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03104     for (iq = 0; iq < ctl->nq; iq++)
03105       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03106               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03107     fprintf(out, "\n");
03108
03109     /* Set geolocation and search radius... */
03110     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03111     rmax2 = SQR(ctl->stat_r);
03112   }
03113
03114   /* Set time interval for output... */
03115   t0 = t - 0.5 * ctl->dt_mod;
03116   t1 = t + 0.5 * ctl->dt_mod;
03117
03118   /* Loop over air parcels... */
03119   for (ip = 0; ip < atm->np; ip++) {
03120
03121     /* Check time... */
03122     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03123       continue;
03124
03125     /* Check station flag... */
03126     if (ctl->qnt_stat >= 0)
03127       if (atm->q[ctl->qnt_stat][ip])
03128         continue;
03129
03130     /* Get Cartesian coordinates... */
```

```
03131     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03132
03133     /* Check horizontal distance... */
03134     if (DIST2(x0, x1) > rmax2)
03135       continue;
03136
03137     /* Set station flag... */
03138     if (ctl->qnt_stat >= 0)
03139       atm->q[ctl->qnt_stat][ip] = 1;
03140
03141     /* Write data... */
03142     fprintf(out, "%.2f %g %g %g",
03143             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
03144     for (iq = 0; iq < ctl->nq; iq++) {
03145       fprintf(out, " ");
03146       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03147     }
03148     fprintf(out, "\n");
03149   }
03150
03151   /* Close file... */
03152   if (t == ctl->t_stop)
03153     fclose(out);
03154 }
```

Here is the call graph for this function:

5.20 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) 2013-2019 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   isec = locate_irr(secs, 12, sec);
00292   ilat = locate_reg(lats, 18, lat);
00293   ip = locate_irr(ps, 10, p);
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;
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   ilat = locate_reg(lats, 73, lat);
00439   imon = locate_irr(doys, 12, doy);
00440
00441   /* Interpolate... */
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   return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447 }
00448
00449 /*****************************************************************************/
00450
00451 void day2doy(
00452   int year,
00453   int mon,
00454   int day,
00455   int *doy) {
00456
00457   int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00458   int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00459
00460   /* Get day of year... */
00461   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00462     *doy = d01[mon - 1] + day - 1;
00463   else
00464     *doy = d0[mon - 1] + day - 1;
00465 }
00466
00467 /*****************************************************************************/
00468
00469 void doy2day(
00470   int year,
00471   int doy,
00472   int *mon,
00473   int *day) {
00474
00475   int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00476   int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
```

```
00477   int i;
00478
00479   /* Get month and day... */
00480   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00481     for (i = 11; i >= 0; i--)
00482       if (d0l[i] <= doy)
00483         break;
00484     *mon = i + 1;
00485     *day = doy - d0l[i] + 1;
00486   } else {
00487     for (i = 11; i >= 0; i--)
00488       if (d0[i] <= doy)
00489         break;
00490     *mon = i + 1;
00491     *day = doy - d0[i] + 1;
00492   }
00493 }
00494
00495 /*****************************************************************************/
00496
00497 void geo2cart(
00498   double z,
00499   double lon,
00500   double lat,
00501   double *x) {
00502
00503   double radius;
00504
00505   radius = z + RE;
00506   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00507   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00508   x[2] = radius * sin(lat / 180 * M_PI);
00509 }
00510
00511 /*****************************************************************************/
00512
00513 void get_met(
00514   ctl_t * ctl,
00515   char *metbase,
00516   double t,
00517   met_t ** met0,
00518   met_t ** met1) {
00519
00520   static int init, ip, ix, iy;
00521
00522   met_t *mets;
00523
00524   char filename[LEN];
00525
00526   /* Init... */
00527   if (t == ctl->t_start || !init) {
00528     init = 1;
00529
00530     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00531     if (!read_met(ctl, filename, *met0))
00532       ERRMSG("Cannot open file!");
00533
00534     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00535     if (!read_met(ctl, filename, *met1))
00536       ERRMSG("Cannot open file!");
00537   }
00538
00539   /* Read new data for forward trajectories... */
00540   if (t > (*met1)->time && ctl->direction == 1) {
00541     mets = *met1;
00542     *met1 = *met0;
00543     *met0 = mets;
00544     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00545     if (!read_met(ctl, filename, *met1))
00546       ERRMSG("Cannot open file!");
00547   }
00548
00549   /* Read new data for backward trajectories... */
00550   if (t < (*met0)->time && ctl->direction == -1) {
00551     mets = *met1;
00552     *met1 = *met0;
00553     *met0 = mets;
00554     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00555     if (!read_met(ctl, filename, *met0))
00556       ERRMSG("Cannot open file!");
00557   }
00558
00559   /* Check that grids are consistent... */
00560   if ((*met0)->nx != (*met1)->nx
00561       || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00562     ERRMSG("Meteo grid dimensions do not match!");
```

```
00563   for (ix = 0; ix < (*met0)->nx; ix++)
00564     if ((*met0)->lon[ix] != (*met1)->lon[ix])
00565       ERRMSG("Meteo grid longitudes do not match!");
00566   for (iy = 0; iy < (*met0)->ny; iy++)
00567     if ((*met0)->lat[iy] != (*met1)->lat[iy])
00568       ERRMSG("Meteo grid latitudes do not match!");
00569   for (ip = 0; ip < (*met0)->np; ip++)
00570     if ((*met0)->p[ip] != (*met1)->p[ip])
00571       ERRMSG("Meteo grid pressure levels do not match!");
00572 }
00573
00574 /*****************************************************************************/
00575
00576 void get_met_help(
00577   double t,
00578   int direct,
00579   char *metbase,
00580   double dt_met,
00581   char *filename) {
00582
00583   char repl[LEN];
00584
00585   double t6, r;
00586
00587   int year, mon, day, hour, min, sec;
00588
00589   /* Round time to fixed intervals... */
00590   if (direct == -1)
00591     t6 = floor(t / dt_met) * dt_met;
00592   else
00593     t6 = ceil(t / dt_met) * dt_met;
00594
00595   /* Decode time... */
00596   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00597
00598   /* Set filename... */
00599   sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
00600   sprintf(repl, "%d", year);
00601   get_met_replace(filename, "YYYY", repl);
00602   sprintf(repl, "%02d", mon);
00603   get_met_replace(filename, "MM", repl);
00604   sprintf(repl, "%02d", day);
00605   get_met_replace(filename, "DD", repl);
00606   sprintf(repl, "%02d", hour);
00607   get_met_replace(filename, "HH", repl);
00608 }
00609
00610 /*****************************************************************************/
00611
00612 void get_met_replace(
00613   char *orig,
00614   char *search,
00615   char *repl) {
00616
00617   char buffer[LEN], *ch;
00618
00619   int i;
00620
00621   /* Iterate... */
00622   for (i = 0; i < 3; i++) {
00623
00624     /* Replace substring... */
00625     if (!(ch = strstr(orig, search)))
00626       return;
00627     strncpy(buffer, orig, (size_t) (ch - orig));
00628     buffer[ch - orig] = 0;
00629     sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
00630     orig[0] = 0;
00631     strcpy(orig, buffer);
00632   }
00633 }
00634
00635 /*****************************************************************************/
00636
00637 void intpol_met_2d(
00638   double array[EX][EY],
00639   int ix,
00640   int iy,
00641   double wx,
00642   double wy,
00643   double *var) {
00644
00645   double aux00, aux01, aux10, aux11;
00646
00647   /* Set variables... */
00648   aux00 = array[ix][iy];
00649   aux01 = array[ix][iy + 1];
```

```
00650   aux10 = array[ix + 1][iy];
00651   aux11 = array[ix + 1][iy + 1];
00652
00653   /* Interpolate horizontally... */
00654   aux00 = wy * (aux00 - aux01) + aux01;
00655   aux11 = wy * (aux10 - aux11) + aux11;
00656   *var = wx * (aux00 - aux11) + aux11;
00657 }
00658
00659 /*****************************************************************************/
00660
00661 void intpol_met_3d(
00662   float array[EX][EY][EP],
00663   int ip,
00664   int ix,
00665   int iy,
00666   double wp,
00667   double wx,
00668   double wy,
00669   double *var) {
00670
00671   double aux00, aux01, aux10, aux11;
00672
00673   /* Interpolate vertically... */
00674   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00675     + array[ix][iy][ip + 1];
00676   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00677     + array[ix][iy + 1][ip + 1];
00678   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00679     + array[ix + 1][iy][ip + 1];
00680   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00681     + array[ix + 1][iy + 1][ip + 1];
00682
00683   /* Interpolate horizontally... */
00684   aux00 = wy * (aux00 - aux01) + aux01;
00685   aux11 = wy * (aux10 - aux11) + aux11;
00686   *var = wx * (aux00 - aux11) + aux11;
00687 }
00688
00689 /*****************************************************************************/
00690
00691 void intpol_met_space(
00692   met_t * met,
00693   double p,
00694   double lon,
00695   double lat,
00696   double *ps,
00697   double *pt,
00698   double *z,
00699   double *t,
00700   double *u,
00701   double *v,
00702   double *w,
00703   double *pv,
00704   double *h2o,
00705   double *o3) {
00706
00707   double wp, wx, wy;
00708
00709   int ip, ix, iy;
00710
00711   /* Check longitude... */
00712   if (met->lon[met->nx - 1] > 180 && lon < 0)
00713     lon += 360;
00714
00715   /* Get indices... */
00716   ip = locate_irr(met->p, met->np, p);
00717   ix = locate_reg(met->lon, met->nx, lon);
00718   iy = locate_reg(met->lat, met->ny, lat);
00719
00720   /* Get weights... */
00721   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00722   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00723   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00724
00725   /* Interpolate... */
00726   if (ps != NULL)
00727     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00728   if (pt != NULL)
00729     intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00730   if (z != NULL)
00731     intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00732   if (t != NULL)
00733     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00734   if (u != NULL)
00735     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00736   if (v != NULL)
```

```
00737     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00738   if (w != NULL)
00739     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00740   if (pv != NULL)
00741     intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00742   if (h2o != NULL)
00743     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00744   if (o3 != NULL)
00745     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00746 }
00747
00748 /*****************************************************************************/
00749
00750 void intpol_met_time(
00751   met_t * met0,
00752   met_t * met1,
00753   double ts,
00754   double p,
00755   double lon,
00756   double lat,
00757   double *ps,
00758   double *pt,
00759   double *z,
00760   double *t,
00761   double *u,
00762   double *v,
00763   double *w,
00764   double *pv,
00765   double *h2o,
00766   double *o3) {
00767
00768   double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00769     v0, v1, w0, w1, wt, z0, z1;
00770
00771   /* Spatial interpolation... */
00772   intpol_met_space(met0, p, lon, lat,
00773                    ps == NULL ? NULL : &ps0,
00774                    pt == NULL ? NULL : &pt0,
00775                    z == NULL ? NULL : &z0,
00776                    t == NULL ? NULL : &t0,
00777                    u == NULL ? NULL : &u0,
00778                    v == NULL ? NULL : &v0,
00779                    w == NULL ? NULL : &w0,
00780                    pv == NULL ? NULL : &pv0,
00781                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00782   intpol_met_space(met1, p, lon, lat,
00783                    ps == NULL ? NULL : &ps1,
00784                    pt == NULL ? NULL : &pt1,
00785                    z == NULL ? NULL : &z1,
00786                    t == NULL ? NULL : &t1,
00787                    u == NULL ? NULL : &u1,
00788                    v == NULL ? NULL : &v1,
00789                    w == NULL ? NULL : &w1,
00790                    pv == NULL ? NULL : &pv1,
00791                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00792
00793   /* Get weighting factor... */
00794   wt = (met1->time - ts) / (met1->time - met0->time);
00795
00796   /* Interpolate... */
00797   if (ps != NULL)
00798     *ps = wt * (ps0 - ps1) + ps1;
00799   if (pt != NULL)
00800     *pt = wt * (pt0 - pt1) + pt1;
00801   if (z != NULL)
00802     *z = wt * (z0 - z1) + z1;
00803   if (t != NULL)
00804     *t = wt * (t0 - t1) + t1;
00805   if (u != NULL)
00806     *u = wt * (u0 - u1) + u1;
00807   if (v != NULL)
00808     *v = wt * (v0 - v1) + v1;
00809   if (w != NULL)
00810     *w = wt * (w0 - w1) + w1;
00811   if (pv != NULL)
00812     *pv = wt * (pv0 - pv1) + pv1;
00813   if (h2o != NULL)
00814     *h2o = wt * (h2o0 - h2o1) + h2o1;
00815   if (o3 != NULL)
00816     *o3 = wt * (o30 - o31) + o31;
00817 }
00818
00819 /*****************************************************************************/
00820
00821 void jsec2time(
00822   double jsec,
00823   int *year,
```

```
00824   int *mon,
00825   int *day,
00826   int *hour,
00827   int *min,
00828   int *sec,
00829   double *remain) {
00830
00831   struct tm t0, *t1;
00832
00833   time_t jsec0;
00834
00835   t0.tm_year = 100;
00836   t0.tm_mon = 0;
00837   t0.tm_mday = 1;
00838   t0.tm_hour = 0;
00839   t0.tm_min = 0;
00840   t0.tm_sec = 0;
00841
00842   jsec0 = (time_t) jsec + timegm(&t0);
00843   t1 = gmtime(&jsec0);
00844
00845   *year = t1->tm_year + 1900;
00846   *mon = t1->tm_mon + 1;
00847   *day = t1->tm_mday;
00848   *hour = t1->tm_hour;
00849   *min = t1->tm_min;
00850   *sec = t1->tm_sec;
00851   *remain = jsec - floor(jsec);
00852 }
00853
00854 /*****************************************************************************/
00855
00856 int locate_irr(
00857   double *xx,
00858   int n,
00859   double x) {
00860
00861   int i, ilo, ihi;
00862
00863   ilo = 0;
00864   ihi = n - 1;
00865   i = (ihi + ilo) >> 1;
00866
00867   if (xx[i] < xx[i + 1])
00868     while (ihi > ilo + 1) {
00869       i = (ihi + ilo) >> 1;
00870       if (xx[i] > x)
00871         ihi = i;
00872       else
00873         ilo = i;
00874   } else
00875     while (ihi > ilo + 1) {
00876       i = (ihi + ilo) >> 1;
00877       if (xx[i] <= x)
00878         ihi = i;
00879       else
00880         ilo = i;
00881     }
00882
00883   return ilo;
00884 }
00885
00886 /*****************************************************************************/
00887
00888 int locate_reg(
00889   double *xx,
00890   int n,
00891   double x) {
00892
00893   int i;
00894
00895   /* Calculate index... */
00896   i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00897
00898   /* Check range... */
00899   if (i < 0)
00900     i = 0;
00901   else if (i >= n - 2)
00902     i = n - 2;
00903
00904   return i;
00905 }
00906
00907 /*****************************************************************************/
00908
00909 int read_atm(
00910   const char *filename,
```

```
00911   ctl_t * ctl,
00912   atm_t * atm) {
00913
00914   FILE *in;
00915
00916   char line[LEN], *tok;
00917
00918   double t0;
00919
00920   int dimid, ip, iq, ncid, varid;
00921
00922   size_t nparts;
00923
00924   /* Init... */
00925   atm->np = 0;
00926
00927   /* Write info... */
00928   printf("Read atmospheric data: %s\n", filename);
00929
00930   /* Read ASCII data... */
00931   if (ctl->atm_type == 0) {
00932
00933     /* Open file... */
00934     if (!(in = fopen(filename, "r")))
00935       return 0;
00936
00937     /* Read line... */
00938     while (fgets(line, LEN, in)) {
00939
00940       /* Read data... */
00941       TOK(line, tok, "%lg", atm->time[atm->np]);
00942       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00943       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00944       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00945       for (iq = 0; iq < ctl->nq; iq++)
00946         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00947
00948       /* Convert altitude to pressure... */
00949       atm->p[atm->np] = P(atm->p[atm->np]);
00950
00951       /* Increment data point counter... */
00952       if ((++atm->np) > NP)
00953         ERRMSG("Too many data points!");
00954     }
00955
00956     /* Close file... */
00957     fclose(in);
00958   }
00959
00960   /* Read binary data... */
00961   else if (ctl->atm_type == 1) {
00962
00963     /* Open file... */
00964     if (!(in = fopen(filename, "r")))
00965       return 0;
00966
00967     /* Read data... */
00968     FREAD(&atm->np, int, 1, in);
00969     FREAD(atm->time, double,
00970           (size_t) atm->np,
00971           in);
00972     FREAD(atm->p, double,
00973           (size_t) atm->np,
00974           in);
00975     FREAD(atm->lon, double,
00976           (size_t) atm->np,
00977           in);
00978     FREAD(atm->lat, double,
00979           (size_t) atm->np,
00980           in);
00981     for (iq = 0; iq < ctl->nq; iq++)
00982       FREAD(atm->q[iq], double,
00983             (size_t) atm->np,
00984             in);
00985
00986     /* Close file... */
00987     fclose(in);
00988   }
00989
00990   /* Read netCDF data... */
00991   else if (ctl->atm_type == 2) {
00992
00993     /* Open file... */
00994     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
00995       return 0;
00996
00997     /* Get dimensions... */
```

```
00998     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
00999     NC(nc_inq_dimlen(ncid, dimid, &nparts));
01000     atm->np = (int) nparts;
01001     if (atm->np > NP)
01002       ERRMSG("Too many particles!");
01003
01004     /* Get time... */
01005     NC(nc_inq_varid(ncid, "time", &varid));
01006     NC(nc_get_var_double(ncid, varid, &t0));
01007     for (ip = 0; ip < atm->np; ip++)
01008       atm->time[ip] = t0;
01009
01010     /* Read geolocations... */
01011     NC(nc_inq_varid(ncid, "PRESS", &varid));
01012     NC(nc_get_var_double(ncid, varid, atm->p));
01013     NC(nc_inq_varid(ncid, "LON", &varid));
01014     NC(nc_get_var_double(ncid, varid, atm->lon));
01015     NC(nc_inq_varid(ncid, "LAT", &varid));
01016     NC(nc_get_var_double(ncid, varid, atm->lat));
01017
01018     /* Read variables... */
01019     if (ctl->qnt_p >= 0)
01020       if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01021         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01022     if (ctl->qnt_t >= 0)
01023       if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01024         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01025     if (ctl->qnt_u >= 0)
01026       if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01027         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
01028     if (ctl->qnt_v >= 0)
01029       if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01030         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01031     if (ctl->qnt_w >= 0)
01032       if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01033         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01034     if (ctl->qnt_h2o >= 0)
01035       if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01036         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01037     if (ctl->qnt_o3 >= 0)
01038       if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
01039         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01040     if (ctl->qnt_theta >= 0)
01041       if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01042         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01043     if (ctl->qnt_pv >= 0)
01044       if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01045         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01046
01047     /* Check data... */
01048     for (ip = 0; ip < atm->np; ip++)
01049       if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01050           || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01051           || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01052           || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01053           || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01054         atm->time[ip] = GSL_NAN;
01055         atm->p[ip] = GSL_NAN;
01056         atm->lon[ip] = GSL_NAN;
01057         atm->lat[ip] = GSL_NAN;
01058         for (iq = 0; iq < ctl->nq; iq++)
01059           atm->q[iq][ip] = GSL_NAN;
01060       } else {
01061         if (ctl->qnt_h2o >= 0)
01062           atm->q[ctl->qnt_h2o][ip] *= 1.608;
01063         if (ctl->qnt_pv >= 0)
01064           atm->q[ctl->qnt_pv][ip] *= 1e6;
01065         if (atm->lon[ip] > 180)
01066           atm->lon[ip] -= 360;
01067       }
01068
01069     /* Close file... */
01070     NC(nc_close(ncid));
01071   }
01072
01073   /* Error... */
01074   else
01075     ERRMSG("Atmospheric data type not supported!");
01076
01077   /* Check number of points... */
01078   if (atm->np < 1)
01079     ERRMSG("Can not read any data!");
01080
01081   /* Return success... */
01082   return 1;
01083 }
01084
```

```
01085 /*****************************************************************************/
01086
01087 void read_ctl(
01088   const char *filename,
01089   int argc,
01090   char *argv[],
01091   ctl_t * ctl) {
01092
01093   int ip, iq;
01094
01095   /* Write info... */
01096   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01097          "(executable: %s | compiled: %s, %s)\n\n",
01098          argv[0], __DATE__, __TIME__);
01099
01100   /* Initialize quantity indices... */
01101   ctl->qnt_ens = -1;
01102   ctl->qnt_m = -1;
01103   ctl->qnt_r = -1;
01104   ctl->qnt_rho = -1;
01105   ctl->qnt_ps = -1;
01106   ctl->qnt_pt = -1;
01107   ctl->qnt_z = -1;
01108   ctl->qnt_p = -1;
01109   ctl->qnt_t = -1;
01110   ctl->qnt_u = -1;
01111   ctl->qnt_v = -1;
01112   ctl->qnt_w = -1;
01113   ctl->qnt_h2o = -1;
01114   ctl->qnt_o3 = -1;
01115   ctl->qnt_theta = -1;
01116   ctl->qnt_vh = -1;
01117   ctl->qnt_vz = -1;
01118   ctl->qnt_pv = -1;
01119   ctl->qnt_tice = -1;
01120   ctl->qnt_tsts = -1;
01121   ctl->qnt_tnat = -1;
01122   ctl->qnt_stat = -1;
01123
01124   /* Read quantities... */
01125   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01126   if (ctl->nq > NQ)
01127     ERRMSG("Too many quantities!");
01128   for (iq = 0; iq < ctl->nq; iq++) {
01129
01130     /* Read quantity name and format... */
01131     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01132     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01133              ctl->qnt_format[iq]);
01134
01135     /* Try to identify quantity... */
01136     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01137       ctl->qnt_ens = iq;
01138       sprintf(ctl->qnt_unit[iq], "-");
01139     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01140       ctl->qnt_m = iq;
01141       sprintf(ctl->qnt_unit[iq], "kg");
01142     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01143       ctl->qnt_r = iq;
01144       sprintf(ctl->qnt_unit[iq], "m");
01145     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01146       ctl->qnt_rho = iq;
01147       sprintf(ctl->qnt_unit[iq], "kg/m^3");
01148     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01149       ctl->qnt_ps = iq;
01150       sprintf(ctl->qnt_unit[iq], "hPa");
01151     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01152       ctl->qnt_pt = iq;
01153       sprintf(ctl->qnt_unit[iq], "hPa");
01154     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01155       ctl->qnt_z = iq;
01156       sprintf(ctl->qnt_unit[iq], "km");
01157     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01158       ctl->qnt_p = iq;
01159       sprintf(ctl->qnt_unit[iq], "hPa");
01160     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01161       ctl->qnt_t = iq;
01162       sprintf(ctl->qnt_unit[iq], "K");
01163     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01164       ctl->qnt_u = iq;
01165       sprintf(ctl->qnt_unit[iq], "m/s");
01166     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01167       ctl->qnt_v = iq;
01168       sprintf(ctl->qnt_unit[iq], "m/s");
01169     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01170       ctl->qnt_w = iq;
01171       sprintf(ctl->qnt_unit[iq], "hPa/s");
```

```
01172     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01173       ctl->qnt_h2o = iq;
01174       sprintf(ctl->qnt_unit[iq], "1");
01175     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01176       ctl->qnt_o3 = iq;
01177       sprintf(ctl->qnt_unit[iq], "1");
01178     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01179       ctl->qnt_theta = iq;
01180       sprintf(ctl->qnt_unit[iq], "K");
01181     } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01182       ctl->qnt_vh = iq;
01183       sprintf(ctl->qnt_unit[iq], "m/s");
01184     } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01185       ctl->qnt_vz = iq;
01186       sprintf(ctl->qnt_unit[iq], "m/s");
01187     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01188       ctl->qnt_pv = iq;
01189       sprintf(ctl->qnt_unit[iq], "PVU");
01190     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01191       ctl->qnt_tice = iq;
01192       sprintf(ctl->qnt_unit[iq], "K");
01193     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01194       ctl->qnt_tsts = iq;
01195       sprintf(ctl->qnt_unit[iq], "K");
01196     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01197       ctl->qnt_tnat = iq;
01198       sprintf(ctl->qnt_unit[iq], "K");
01199     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01200       ctl->qnt_stat = iq;
01201       sprintf(ctl->qnt_unit[iq], "-");
01202     } else
01203       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01204   }
01205
01206   /* Time steps of simulation... */
01207   ctl->direction =
01208     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01209   if (ctl->direction != -1 && ctl->direction != 1)
01210     ERRMSG("Set DIRECTION to -1 or 1!");
01211   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
01212   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01213
01214   /* Meteorological data... */
01215   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01216   ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01217   ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01218   ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01219   ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01220   ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01221   ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01222   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01223   if (ctl->met_np > EP)
01224     ERRMSG("Too many levels!");
01225   for (ip = 0; ip < ctl->met_np; ip++)
01226     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01227   ctl->met_tropo
01228     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01229   scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01230   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01231   ctl->met_dt_out =
01232     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01233
01234   /* Isosurface parameters... */
01235   ctl->isosurf
01236     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01237   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01238
01239   /* Diffusion parameters... */
01240   ctl->turb_dx_trop
01241     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01242   ctl->turb_dx_strat
01243     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01244   ctl->turb_dz_trop
01245     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01246   ctl->turb_dz_strat
01247     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01248   ctl->turb_mesox =
01249     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01250   ctl->turb_mesoz =
01251     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01252
01253   /* Mass and life time... */
01254   ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
01255   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01256   ctl->tdec_strat =
01257     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01258
```

```
01259   /* PSC analysis... */
01260   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01261   ctl->psc_hno3 =
01262     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01263
01264   /* Output of atmospheric data... */
01265   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
01266   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01267   ctl->atm_dt_out =
01268     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01269   ctl->atm_filter =
01270     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01271   ctl->atm_type =
01272     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01273
01274   /* Output of CSI data... */
01275   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01276   ctl->csi_dt_out =
01277     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01278   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
      csi_obsfile);
01279   ctl->csi_obsmin =
01280     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01281   ctl->csi_modmin =
01282     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01283   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01284   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01285   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01286   ctl->csi_lon0 =
01287     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01288   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01289   ctl->csi_nx =
01290     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01291   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01292   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01293   ctl->csi_ny =
01294     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01295
01296   /* Output of ensemble data... */
01297   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01298
01299   /* Output of grid data... */
01300   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01301            ctl->grid_basename);
01302   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01303   ctl->grid_dt_out =
01304     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01305   ctl->grid_sparse =
01306     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01307   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01308   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01309   ctl->grid_nz =
01310     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01311   ctl->grid_lon0 =
01312     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01313   ctl->grid_lon1 =
01314     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01315   ctl->grid_nx =
01316     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01317   ctl->grid_lat0 =
01318     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01319   ctl->grid_lat1 =
01320     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01321   ctl->grid_ny =
01322     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01323
01324   /* Output of profile data... */
01325   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01326            ctl->prof_basename);
01327   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
01328   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01329   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01330   ctl->prof_nz =
01331     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01332   ctl->prof_lon0 =
01333     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01334   ctl->prof_lon1 =
01335     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
01336   ctl->prof_nx =
01337     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01338   ctl->prof_lat0 =
01339     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
```

```
01340   ctl->prof_lat1 =
01341     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01342   ctl->prof_ny =
01343     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01344
01345   /* Output of station data... */
01346   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01347            ctl->stat_basename);
01348   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01349   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01350   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01351 }
01352
01353 /*****************************************************************************/
01354
01355 int read_met(
01356   ctl_t * ctl,
01357   char *filename,
01358   met_t * met) {
01359
01360   char cmd[2 * LEN], levname[LEN], tstr[10];
01361
01362   float help[EX * EY];
01363
01364   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01365
01366   size_t np, nx, ny;
01367
01368   /* Write info... */
01369   printf("Read meteorological data: %s\n", filename);
01370
01371   /* Open netCDF file... */
01372   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01373
01374     /* Try to stage meteo file... */
01375     if (ctl->met_stage[0] != '-') {
01376       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01377               year, mon, day, hour, filename);
01378       if (system(cmd) != 0)
01379         ERRMSG("Error while staging meteo data!");
01380     }
01381
01382     /* Try to open again... */
01383     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
01384       return 0;
01385   }
01386
01387   /* Get time from filename... */
01388   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01389   year = atoi(tstr);
01390   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01391   mon = atoi(tstr);
01392   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01393   day = atoi(tstr);
01394   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
01395   hour = atoi(tstr);
01396   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01397
01398   /* Get dimensions... */
01399   NC(nc_inq_dimid(ncid, "lon", &dimid));
01400   NC(nc_inq_dimlen(ncid, dimid, &nx));
01401   if (nx < 2 || nx > EX)
01402     ERRMSG("Number of longitudes out of range!");
01403
01404   NC(nc_inq_dimid(ncid, "lat", &dimid));
01405   NC(nc_inq_dimlen(ncid, dimid, &ny));
01406   if (ny < 2 || ny > EY)
01407     ERRMSG("Number of latitudes out of range!");
01408
01409   sprintf(levname, "lev");
01410   NC(nc_inq_dimid(ncid, levname, &dimid));
01411   NC(nc_inq_dimlen(ncid, dimid, &np));
01412   if (np == 1) {
01413     sprintf(levname, "lev_2");
01414     NC(nc_inq_dimid(ncid, levname, &dimid));
01415     NC(nc_inq_dimlen(ncid, dimid, &np));
01416   }
01417   if (np < 2 || np > EP)
01418     ERRMSG("Number of levels out of range!");
01419
01420   /* Store dimensions... */
01421   met->np = (int) np;
01422   met->nx = (int) nx;
01423   met->ny = (int) ny;
01424
01425   /* Get horizontal grid... */
01426   NC(nc_inq_varid(ncid, "lon", &varid));
```

```
01427   NC(nc_get_var_double(ncid, varid, met->lon));
01428   NC(nc_inq_varid(ncid, "lat", &varid));
01429   NC(nc_get_var_double(ncid, varid, met->lat));
01430
01431   /* Read meteorological data... */
01432   read_met_help(ncid, "t", "T", met, met->t, 1.0);
01433   read_met_help(ncid, "u", "U", met, met->u, 1.0);
01434   read_met_help(ncid, "v", "V", met, met->v, 1.0);
01435   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01436   read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
01437   read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));
01438
01439   /* Meteo data on pressure levels... */
01440   if (ctl->met_np <= 0) {
01441
01442     /* Read pressure levels from file... */
01443     NC(nc_inq_varid(ncid, levname, &varid));
01444     NC(nc_get_var_double(ncid, varid, met->p));
01445     for (ip = 0; ip < met->np; ip++)
01446       met->p[ip] /= 100.;
01447
01448     /* Extrapolate data for lower boundary... */
01449     read_met_extrapolate(met);
01450   }
01451
01452   /* Meteo data on model levels... */
01453   else {
01454
01455     /* Read pressure data from file... */
01456     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01457
01458     /* Interpolate from model levels to pressure levels... */
01459     read_met_ml2pl(ctl, met, met->t);
01460     read_met_ml2pl(ctl, met, met->u);
01461     read_met_ml2pl(ctl, met, met->v);
01462     read_met_ml2pl(ctl, met, met->w);
01463     read_met_ml2pl(ctl, met, met->h2o);
01464     read_met_ml2pl(ctl, met, met->o3);
01465
01466     /* Set pressure levels... */
01467     met->np = ctl->met_np;
01468     for (ip = 0; ip < met->np; ip++)
01469       met->p[ip] = ctl->met_p[ip];
01470   }
01471
01472   /* Check ordering of pressure levels... */
01473   for (ip = 1; ip < met->np; ip++)
01474     if (met->p[ip - 1] < met->p[ip])
01475       ERRMSG("Pressure levels must be descending!");
01476
01477   /* Read surface pressure... */
01478   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01479       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01480     NC(nc_get_var_float(ncid, varid, help));
01481     for (iy = 0; iy < met->ny; iy++)
01482       for (ix = 0; ix < met->nx; ix++)
01483         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01484   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01485              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01486     NC(nc_get_var_float(ncid, varid, help));
01487     for (iy = 0; iy < met->ny; iy++)
01488       for (ix = 0; ix < met->nx; ix++)
01489         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01490   } else
01491     for (ix = 0; ix < met->nx; ix++)
01492       for (iy = 0; iy < met->ny; iy++)
01493         met->ps[ix][iy] = met->p[0];
01494
01495   /* Create periodic boundary conditions... */
01496   read_met_periodic(met);
01497
01498   /* Calculate geopotential heights... */
01499   read_met_geopot(ctl, met);
01500
01501   /* Calculate potential vorticity... */
01502   read_met_pv(met);
01503
01504   /* Calculate tropopause pressure... */
01505   read_met_tropo(ctl, met);
01506
01507   /* Downsampling... */
01508   read_met_sample(ctl, met);
01509
01510   /* Close file... */
01511   NC(nc_close(ncid));
01512
01513   /* Return success... */
```

```
01514   return 1;
01515 }
01516
01517 /*****************************************************************************/
01518
01519 void read_met_extrapolate(
01520   met_t * met) {
01521
01522   int ip, ip0, ix, iy;
01523
01524   /* Loop over columns... */
01525 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01526   for (ix = 0; ix < met->nx; ix++)
01527     for (iy = 0; iy < met->ny; iy++) {
01528
01529       /* Find lowest valid data point... */
01530       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01531         if (!gsl_finite(met->t[ix][iy][ip0])
01532             || !gsl_finite(met->u[ix][iy][ip0])
01533             || !gsl_finite(met->v[ix][iy][ip0])
01534             || !gsl_finite(met->w[ix][iy][ip0]))
01535           break;
01536
01537       /* Extrapolate... */
01538       for (ip = ip0; ip >= 0; ip--) {
01539         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01540         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01541         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01542         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01543         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01544         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01545       }
01546     }
01547 }
01548
01549 /*****************************************************************************/
01550
01551 void read_met_geopot(
01552   ctl_t * ctl,
01553   met_t * met) {
01554
01555   static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01556
01557   static int init, topo_nx = -1, topo_ny;
01558
01559   FILE *in;
01560
01561   char line[LEN];
01562
01563   double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01564
01565   float help[EX][EY];
01566
01567   int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01568
01569   /* Initialize geopotential heights... */
01570 #pragma omp parallel for default(shared) private(ix,iy,ip)
01571   for (ix = 0; ix < met->nx; ix++)
01572     for (iy = 0; iy < met->ny; iy++)
01573       for (ip = 0; ip < met->np; ip++)
01574         met->z[ix][iy][ip] = GSL_NAN;
01575
01576   /* Check filename... */
01577   if (ctl->met_geopot[0] == '-')
01578     return;
01579
01580   /* Read surface geopotential... */
01581   if (!init) {
01582     init = 1;
01583
01584     /* Write info... */
01585     printf("Read surface geopotential: %s\n", ctl->met_geopot);
01586
01587     /* Open file... */
01588     if (!(in = fopen(ctl->met_geopot, "r")))
01589       ERRMSG("Cannot open file!");
01590
01591     /* Read data... */
01592     while (fgets(line, LEN, in))
01593       if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01594         if (rlon != rlon_old) {
01595           if ((++topo_nx) >= EX)
01596             ERRMSG("Too many longitudes!");
01597           topo_ny = 0;
01598         }
01599         rlon_old = rlon;
01600         topo_lon[topo_nx] = rlon;
```

```
01601         topo_lat[topo_ny] = rlat;
01602         topo_z[topo_nx][topo_ny] = rz;
01603         if ((++topo_ny) >= EY)
01604           ERRMSG("Too many latitudes!");
01605       }
01606     if ((++topo_nx) >= EX)
01607       ERRMSG("Too many longitudes!");
01608
01609     /* Close file... */
01610     fclose(in);
01611
01612     /* Check grid spacing... */
01613     if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01614         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01615       printf("Warning: Grid spacing does not match!\n");
01616   }
01617
01618   /* Apply hydrostatic equation to calculate geopotential heights... */
01619 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01620   for (ix = 0; ix < met->nx; ix++)
01621     for (iy = 0; iy < met->ny; iy++) {
01622
01623       /* Get surface height... */
01624       lon = met->lon[ix];
01625       if (lon < topo_lon[0])
01626         lon += 360;
01627       else if (lon > topo_lon[topo_nx - 1])
01628         lon -= 360;
01629       lat = met->lat[iy];
01630       tx = locate_reg(topo_lon, topo_nx, lon);
01631       ty = locate_reg(topo_lat, topo_ny, lat);
01632       z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01633                topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01634       z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01635                topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01636       z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01637
01638       /* Find surface pressure level... */
01639       ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01640
01641       /* Get surface temperature... */
01642       ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
01643                met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
01644
01645       /* Upper part of profile... */
01646       met->z[ix][iy][ip0 + 1]
01647         = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01648                    * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01649       for (ip = ip0 + 2; ip < met->np; ip++)
01650         met->z[ix][iy][ip]
01651           = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
01652                      * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01653                      * log(met->p[ip - 1] / met->p[ip]));
01654     }
01655
01656   /* Smooth fields... */
01657   for (ip = 0; ip < met->np; ip++) {
01658
01659     /* Median filter... */
01660 #pragma omp parallel for default(shared) private(ix,iy,n,ix2,ix3,iy2,data)
01661     for (ix = 0; ix < met->nx; ix++)
01662       for (iy = 0; iy < met->ny; iy++) {
01663         n = 0;
01664         for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
01665           ix3 = ix2;
01666           if (ix3 < 0)
01667             ix3 += met->nx;
01668           if (ix3 >= met->nx)
01669             ix3 -= met->nx;
01670           for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01671                iy2++)
01672             if (gsl_finite(met->z[ix3][iy2][ip])) {
01673               data[n] = met->z[ix3][iy2][ip];
01674               n++;
01675             }
01676         }
01677         if (n > 0) {
01678           gsl_sort(data, 1, (size_t) n);
01679           help[ix][iy] = (float)
01680             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01681         } else
01682           help[ix][iy] = GSL_NAN;
01683       }
01684
01685     /* Copy data... */
01686 #pragma omp parallel for default(shared) private(ix,iy)
01687     for (ix = 0; ix < met->nx; ix++)
```

```
01688       for (iy = 0; iy < met->ny; iy++)
01689         met->z[ix][iy][ip] = help[ix][iy];
01690   }
01691 }
01692
01693 /*****************************************************************************/
01694
01695 void read_met_help(
01696   int ncid,
01697   char *varname,
01698   char *varname2,
01699   met_t * met,
01700   float dest[EX][EY][EP],
01701   float scl) {
01702
01703   float *help;
01704
01705   int ip, ix, iy, varid;
01706
01707   /* Check if variable exists... */
01708   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01709     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01710       return;
01711
01712   /* Allocate... */
01713   ALLOC(help, float, met->nx * met->ny * met->np);
01714
01715   /* Read data... */
01716   NC(nc_get_var_float(ncid, varid, help));
01717
01718   /* Copy and check data... */
01719 #pragma omp parallel for default(shared) private(ix,iy,ip)
01720   for (ix = 0; ix < met->nx; ix++)
01721     for (iy = 0; iy < met->ny; iy++)
01722       for (ip = 0; ip < met->np; ip++) {
01723         dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01724         if (fabsf(dest[ix][iy][ip]) < 1e14f)
01725           dest[ix][iy][ip] *= scl;
01726         else
01727           dest[ix][iy][ip] = GSL_NAN;
01728       }
01729
01730   /* Free... */
01731   free(help);
01732 }
01733
01734 /*****************************************************************************/
01735
01736 void read_met_ml2pl(
01737   ctl_t * ctl,
01738   met_t * met,
01739   float var[EX][EY][EP]) {
01740
01741   double aux[EP], p[EP], pt;
01742
01743   int ip, ip2, ix, iy;
01744
01745   /* Loop over columns... */
01746 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01747   for (ix = 0; ix < met->nx; ix++)
01748     for (iy = 0; iy < met->ny; iy++) {
01749
01750       /* Copy pressure profile... */
01751       for (ip = 0; ip < met->np; ip++)
01752         p[ip] = met->pl[ix][iy][ip];
01753
01754       /* Interpolate... */
01755       for (ip = 0; ip < ctl->met_np; ip++) {
01756         pt = ctl->met_p[ip];
01757         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01758           pt = p[0];
01759         else if ((pt > p[met->np - 1] && p[1] > p[0])
01760                  || (pt < p[met->np - 1] && p[1] < p[0]))
01761           pt = p[met->np - 1];
01762         ip2 = locate_irr(p, met->np, pt);
01763         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01764                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01765       }
01766
01767       /* Copy data... */
01768       for (ip = 0; ip < ctl->met_np; ip++)
01769         var[ix][iy][ip] = (float) aux[ip];
01770     }
01771 }
01772
01773 /*****************************************************************************/
01774
```

```
01775 void read_met_periodic(
01776   met_t * met) {
01777
01778   int ip, iy;
01779
01780   /* Check longitudes... */
01781   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01782              + met->lon[1] - met->lon[0] - 360) < 0.01))
01783     return;
01784
01785   /* Increase longitude counter... */
01786   if ((++met->nx) > EX)
01787     ERRMSG("Cannot create periodic boundary conditions!");
01788
01789   /* Set longitude... */
01790   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01791
01792   /* Loop over latitudes and pressure levels... */
01793 #pragma omp parallel for default(shared) private(iy,ip)
01794   for (iy = 0; iy < met->ny; iy++) {
01795     met->ps[met->nx - 1][iy] = met->ps[0][iy];
01796     met->pt[met->nx - 1][iy] = met->pt[0][iy];
01797     for (ip = 0; ip < met->np; ip++) {
01798       met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01799       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01800       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01801       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01802       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01803       met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01804       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01805       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01806     }
01807   }
01808 }
01809
01810 /*****************************************************************************/
01811
01812 void read_met_pv(
01813   met_t * met) {
01814
01815   double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01816     dtdp, dudp, dvdp, latr, vort, pows[EP];
01817
01818   int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01819
01820   /* Set powers... */
01821   for (ip = 0; ip < met->np; ip++)
01822     pows[ip] = pow(1000. / met->p[ip], 0.286);
01823
01824   /* Loop over grid points... */
01825 #pragma omp parallel for default(shared)
       private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
01826   for (ix = 0; ix < met->nx; ix++) {
01827
01828     /* Set indices... */
01829     ix0 = GSL_MAX(ix - 1, 0);
01830     ix1 = GSL_MIN(ix + 1, met->nx - 1);
01831
01832     /* Loop over grid points... */
01833     for (iy = 0; iy < met->ny; iy++) {
01834
01835       /* Set indices... */
01836       iy0 = GSL_MAX(iy - 1, 0);
01837       iy1 = GSL_MIN(iy + 1, met->ny - 1);
01838
01839       /* Set auxiliary variables... */
01840       latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01841       dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
01842       dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01843       c0 = cos(met->lat[iy0] / 180. * M_PI);
01844       c1 = cos(met->lat[iy1] / 180. * M_PI);
01845       cr = cos(latr / 180. * M_PI);
01846       vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01847
01848       /* Loop over grid points... */
01849       for (ip = 0; ip < met->np; ip++) {
01850
01851         /* Get gradients in longitude... */
01852         dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
01853         dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01854
01855         /* Get gradients in latitude... */
01856         dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01857         dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01858
01859         /* Set indices... */
```

```
01860         ip0 = GSL_MAX(ip - 1, 0);
01861         ip1 = GSL_MIN(ip + 1, met->np - 1);
01862
01863         /* Get gradients in pressure... */
01864         dp0 = 100. * (met->p[ip] - met->p[ip0]);
01865         dp1 = 100. * (met->p[ip1] - met->p[ip]);
01866         if (ip != ip0 && ip != ip1) {
01867           denom = dp0 * dp1 * (dp0 + dp1);
01868           dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01869                   - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01870                   + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01871             / denom;
01872           dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01873                   - dp1 * dp1 * met->u[ix][iy][ip0]
01874                   + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01875             / denom;
01876           dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
01877                   - dp1 * dp1 * met->v[ix][iy][ip0]
01878                   + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01879             / denom;
01880         } else {
01881           denom = dp0 + dp1;
01882           dtdp =
01883             (met->t[ix][iy][ip1] * pows[ip1] -
01884              met->t[ix][iy][ip0] * pows[ip0]) / denom;
01885           dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01886           dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01887         }
01888
01889         /* Calculate PV... */
01890         met->pv[ix][iy][ip] = (float)
01891           (1e6 * G0 *
01892            (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01893       }
01894     }
01895   }
01896 }
01897
01898 /*****************************************************************************/
01899
01900 void read_met_sample(
01901   ctl_t * ctl,
01902   met_t * met) {
01903
01904   met_t *help;
01905
01906   float w, wsum;
01907
01908   int ip, ip2, ix, ix2, ix3, iy, iy2;
01909
01910   /* Check parameters... */
01911   if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
01912       && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01913     return;
01914
01915   /* Allocate... */
01916   ALLOC(help, met_t, 1);
01917
01918   /* Copy data... */
01919   help->nx = met->nx;
01920   help->ny = met->ny;
01921   help->np = met->np;
01922   memcpy(help->lon, met->lon, sizeof(met->lon));
01923   memcpy(help->lat, met->lat, sizeof(met->lat));
01924   memcpy(help->p, met->p, sizeof(met->p));
01925
01926   /* Smoothing... */
01927   for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01928     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01929       for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01930         help->ps[ix][iy] = 0;
01931         help->pt[ix][iy] = 0;
01932         help->z[ix][iy][ip] = 0;
01933         help->t[ix][iy][ip] = 0;
01934         help->u[ix][iy][ip] = 0;
01935         help->v[ix][iy][ip] = 0;
01936         help->w[ix][iy][ip] = 0;
01937         help->pv[ix][iy][ip] = 0;
01938         help->h2o[ix][iy][ip] = 0;
01939         help->o3[ix][iy][ip] = 0;
01940         wsum = 0;
01941         for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
01942           ix3 = ix2;
01943           if (ix3 < 0)
01944             ix3 += met->nx;
01945           else if (ix3 >= met->nx)
01946             ix3 -= met->nx;
```

```
01947
01948           for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01949                iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
01950             for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
01951                  ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
01952               w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
01953                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
01954                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01955               help->ps[ix][iy] += w * met->ps[ix3][iy2];
01956               help->pt[ix][iy] += w * met->pt[ix3][iy2];
01957               help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
01958               help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
01959               help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
01960               help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
01961               help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
01962               help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
01963               help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
01964               help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
01965               wsum += w;
01966             }
01967         }
01968         help->ps[ix][iy] /= wsum;
01969         help->pt[ix][iy] /= wsum;
01970         help->t[ix][iy][ip] /= wsum;
01971         help->z[ix][iy][ip] /= wsum;
01972         help->u[ix][iy][ip] /= wsum;
01973         help->v[ix][iy][ip] /= wsum;
01974         help->w[ix][iy][ip] /= wsum;
01975         help->pv[ix][iy][ip] /= wsum;
01976         help->h2o[ix][iy][ip] /= wsum;
01977         help->o3[ix][iy][ip] /= wsum;
01978       }
01979     }
01980   }
01981
01982   /* Downsampling... */
01983   met->nx = 0;
01984   for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01985     met->lon[met->nx] = help->lon[ix];
01986     met->ny = 0;
01987     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01988       met->lat[met->ny] = help->lat[iy];
01989       met->ps[met->nx][met->ny] = help->ps[ix][iy];
01990       met->pt[met->nx][met->ny] = help->pt[ix][iy];
01991       met->np = 0;
01992       for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01993         met->p[met->np] = help->p[ip];
01994         met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01995         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01996         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01997         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01998         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01999         met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
02000         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
02001         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
02002         met->np++;
02003       }
02004       met->ny++;
02005     }
02006     met->nx++;
02007   }
02008
02009   /* Free... */
02010   free(help);
02011 }
02012
02013 /*****************************************************************************/
02014
02015 void read_met_tropo(
02016   ctl_t * ctl,
02017   met_t * met) {
02018
02019   gsl_interp_accel *acc;
02020
02021   gsl_spline *spline;
02022
02023   double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
02024     z[400], z2[400];
02025
02026   int found, ix, iy, iz, iz2;
02027
02028   /* Allocate... */
02029   acc = gsl_interp_accel_alloc();
02030   spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
02031
02032   /* Get altitude and pressure profiles... */
02033   for (iz = 0; iz < met->np; iz++)
```

```
02034     z[iz] = Z(met->p[iz]);
02035   for (iz = 0; iz <= 170; iz++) {
02036     z2[iz] = 4.5 + 0.1 * iz;
02037     p2[iz] = P(z2[iz]);
02038   }
02039
02040   /* Do not calculate tropopause... */
02041   if (ctl->met_tropo == 0)
02042     for (ix = 0; ix < met->nx; ix++)
02043       for (iy = 0; iy < met->ny; iy++)
02044         met->pt[ix][iy] = GSL_NAN;
02045
02046   /* Use tropopause climatology... */
02047   else if (ctl->met_tropo == 1)
02048     for (ix = 0; ix < met->nx; ix++)
02049       for (iy = 0; iy < met->ny; iy++)
02050         met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
02051
02052   /* Use cold point... */
02053   else if (ctl->met_tropo == 2) {
02054
02055     /* Loop over grid points... */
02056     for (ix = 0; ix < met->nx; ix++)
02057       for (iy = 0; iy < met->ny; iy++) {
02058
02059         /* Interpolate temperature profile... */
02060         for (iz = 0; iz < met->np; iz++)
02061           t[iz] = met->t[ix][iy][iz];
02062         gsl_spline_init(spline, z, t, (size_t) met->np);
02063         for (iz = 0; iz <= 170; iz++)
02064           t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02065
02066         /* Find minimum... */
02067         iz = (int) gsl_stats_min_index(t2, 1, 171);
02068         if (iz <= 0 || iz >= 170)
02069           met->pt[ix][iy] = GSL_NAN;
02070         else
02071           met->pt[ix][iy] = p2[iz];
02072       }
02073   }
02074
02075   /* Use WMO definition... */
02076   else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02077
02078     /* Loop over grid points... */
02079     for (ix = 0; ix < met->nx; ix++)
02080       for (iy = 0; iy < met->ny; iy++) {
02081
02082         /* Interpolate temperature profile... */
02083         for (iz = 0; iz < met->np; iz++)
02084           t[iz] = met->t[ix][iy][iz];
02085         gsl_spline_init(spline, z, t, (size_t) met->np);
02086         for (iz = 0; iz <= 160; iz++)
02087           t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02088
02089         /* Find 1st tropopause... */
02090         met->pt[ix][iy] = GSL_NAN;
02091         for (iz = 0; iz <= 140; iz++) {
02092           found = 1;
02093           for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02094             if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02095                 / log(p2[iz2] / p2[iz]) > 2.0) {
02096               found = 0;
02097               break;
02098             }
02099           if (found) {
02100             if (iz > 0 && iz < 140)
02101               met->pt[ix][iy] = p2[iz];
02102             break;
02103           }
02104         }
02105
02106         /* Find 2nd tropopause... */
02107         if (ctl->met_tropo == 4) {
02108           met->pt[ix][iy] = GSL_NAN;
02109           for (; iz <= 140; iz++) {
02110             found = 1;
02111             for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
02112               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02113                   / log(p2[iz2] / p2[iz]) < 3.0) {
02114                 found = 0;
02115                 break;
02116               }
02117             if (found)
02118               break;
02119           }
02120           for (; iz <= 140; iz++) {
```

```
02121             found = 1;
02122             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02123               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02124                   / log(p2[iz2] / p2[iz]) > 2.0) {
02125                 found = 0;
02126                 break;
02127               }
02128             if (found) {
02129               if (iz > 0 && iz < 140)
02130                 met->pt[ix][iy] = p2[iz];
02131               break;
02132             }
02133           }
02134         }
02135       }
02136   }
02137
02138   /* Use dynamical tropopause... */
02139   else if (ctl->met_tropo == 5) {
02140
02141     /* Loop over grid points... */
02142     for (ix = 0; ix < met->nx; ix++)
02143       for (iy = 0; iy < met->ny; iy++) {
02144
02145         /* Interpolate potential vorticity profile... */
02146         for (iz = 0; iz < met->np; iz++)
02147           pv[iz] = met->pv[ix][iy][iz];
02148         gsl_spline_init(spline, z, pv, (size_t) met->np);
02149         for (iz = 0; iz <= 160; iz++)
02150           pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02151
02152         /* Interpolate potential temperature profile... */
02153         for (iz = 0; iz < met->np; iz++)
02154           th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02155         gsl_spline_init(spline, z, th, (size_t) met->np);
02156         for (iz = 0; iz <= 160; iz++)
02157           th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02158
02159         /* Find dynamical tropopause 3.5 PVU + 380 K */
02160         met->pt[ix][iy] = GSL_NAN;
02161         for (iz = 0; iz <= 160; iz++)
02162           if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02163             if (iz > 0 && iz < 160)
02164               met->pt[ix][iy] = p2[iz];
02165             break;
02166           }
02167       }
02168   }
02169
02170   else
02171     ERRMSG("Cannot calculate tropopause!");
02172
02173   /* Free... */
02174   gsl_spline_free(spline);
02175   gsl_interp_accel_free(acc);
02176 }
02177
02178 /*****************************************************************************/
02179
02180 double scan_ctl(
02181   const char *filename,
02182   int argc,
02183   char *argv[],
02184   const char *varname,
02185   int arridx,
02186   const char *defvalue,
02187   char *value) {
02188
02189   FILE *in = NULL;
02190
02191   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02192     msg[2 * LEN], rvarname[LEN], rval[LEN];
02193
02194   int contain = 0, i;
02195
02196   /* Open file... */
02197   if (filename[strlen(filename) - 1] != '-')
02198     if (!(in = fopen(filename, "r")))
02199       ERRMSG("Cannot open file!");
02200
02201   /* Set full variable name... */
02202   if (arridx >= 0) {
02203     sprintf(fullname1, "%s[%d]", varname, arridx);
02204     sprintf(fullname2, "%s[*]", varname);
02205   } else {
02206     sprintf(fullname1, "%s", varname);
02207     sprintf(fullname2, "%s", varname);
```

```
02208   }
02209
02210   /* Read data... */
02211   if (in != NULL)
02212     while (fgets(line, LEN, in))
02213       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02214         if (strcasecmp(rvarname, fullname1) == 0 ||
02215             strcasecmp(rvarname, fullname2) == 0) {
02216           contain = 1;
02217           break;
02218         }
02219   for (i = 1; i < argc - 1; i++)
02220     if (strcasecmp(argv[i], fullname1) == 0 ||
02221         strcasecmp(argv[i], fullname2) == 0) {
02222       sprintf(rval, "%s", argv[i + 1]);
02223       contain = 1;
02224       break;
02225     }
02226
02227   /* Close file... */
02228   if (in != NULL)
02229     fclose(in);
02230
02231   /* Check for missing variables... */
02232   if (!contain) {
02233     if (strlen(defvalue) > 0)
02234       sprintf(rval, "%s", defvalue);
02235     else {
02236       sprintf(msg, "Missing variable %s!\n", fullname1);
02237       ERRMSG(msg);
02238     }
02239   }
02240
02241   /* Write info... */
02242   printf("%s = %s\n", fullname1, rval);
02243
02244   /* Return values... */
02245   if (value != NULL)
02246     sprintf(value, "%s", rval);
02247   return atof(rval);
02248 }
02249
02250 /*****************************************************************************/
02251
02252 void time2jsec(
02253   int year,
02254   int mon,
02255   int day,
02256   int hour,
02257   int min,
02258   int sec,
02259   double remain,
02260   double *jsec) {
02261
02262   struct tm t0, t1;
02263
02264   t0.tm_year = 100;
02265   t0.tm_mon = 0;
02266   t0.tm_mday = 1;
02267   t0.tm_hour = 0;
02268   t0.tm_min = 0;
02269   t0.tm_sec = 0;
02270
02271   t1.tm_year = year - 1900;
02272   t1.tm_mon = mon - 1;
02273   t1.tm_mday = day;
02274   t1.tm_hour = hour;
02275   t1.tm_min = min;
02276   t1.tm_sec = sec;
02277
02278   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02279 }
02280
02281 /*****************************************************************************/
02282
02283 void timer(
02284   const char *name,
02285   int id,
02286   int mode) {
02287
02288   static double starttime[NTIMER], runtime[NTIMER];
02289
02290   /* Check id... */
02291   if (id < 0 || id >= NTIMER)
02292     ERRMSG("Too many timers!");
02293
02294   /* Start timer... */
```

```
02295   if (mode == 1) {
02296     if (starttime[id] <= 0)
02297       starttime[id] = omp_get_wtime();
02298     else
02299       ERRMSG("Timer already started!");
02300   }
02301
02302   /* Stop timer... */
02303   else if (mode == 2) {
02304     if (starttime[id] > 0) {
02305       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02306       starttime[id] = -1;
02307     }
02308   }
02309
02310   /* Print timer... */
02311   else if (mode == 3) {
02312     printf("%s = %.3f s\n", name, runtime[id]);
02313     runtime[id] = 0;
02314   }
02315 }
02316
02317 /*****************************************************************************/
02318
02319 void write_atm(
02320   const char *filename,
02321   ctl_t * ctl,
02322   atm_t * atm,
02323   double t) {
02324
02325   FILE *in, *out;
02326
02327   char line[LEN];
02328
02329   double r, t0, t1;
02330
02331   int ip, iq, year, mon, day, hour, min, sec;
02332
02333   /* Set time interval for output... */
02334   t0 = t - 0.5 * ctl->dt_mod;
02335   t1 = t + 0.5 * ctl->dt_mod;
02336
02337   /* Write info... */
02338   printf("Write atmospheric data: %s\n", filename);
02339
02340   /* Write ASCII data... */
02341   if (ctl->atm_type == 0) {
02342
02343     /* Check if gnuplot output is requested... */
02344     if (ctl->atm_gpfile[0] != '-') {
02345
02346       /* Create gnuplot pipe... */
02347       if (!(out = popen("gnuplot", "w")))
02348         ERRMSG("Cannot create pipe to gnuplot!");
02349
02350       /* Set plot filename... */
02351       fprintf(out, "set out \"%s.png\"\n", filename);
02352
02353       /* Set time string... */
02354       jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02355       fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02356               year, mon, day, hour, min);
02357
02358       /* Dump gnuplot file to pipe... */
02359       if (!(in = fopen(ctl->atm_gpfile, "r")))
02360         ERRMSG("Cannot open file!");
02361       while (fgets(line, LEN, in))
02362         fprintf(out, "%s", line);
02363       fclose(in);
02364     }
02365
02366     else {
02367
02368       /* Create file... */
02369       if (!(out = fopen(filename, "w")))
02370         ERRMSG("Cannot create file!");
02371     }
02372
02373     /* Write header... */
02374     fprintf(out,
02375             "# $1 = time [s]\n"
02376             "# $2 = altitude [km]\n"
02377             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02378     for (iq = 0; iq < ctl->nq; iq++)
02379       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02380               ctl->qnt_unit[iq]);
02381     fprintf(out, "\n");
```

```
02382
02383     /* Write data... */
02384     for (ip = 0; ip < atm->np; ip++) {
02385
02386       /* Check time... */
02387       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02388         continue;
02389
02390       /* Write output... */
02391       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02392               atm->lon[ip], atm->lat[ip]);
02393       for (iq = 0; iq < ctl->nq; iq++) {
02394         fprintf(out, " ");
02395         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02396       }
02397       fprintf(out, "\n");
02398     }
02399
02400     /* Close file... */
02401     fclose(out);
02402   }
02403
02404   /* Write binary data... */
02405   else if (ctl->atm_type == 1) {
02406
02407     /* Create file... */
02408     if (!(out = fopen(filename, "w")))
02409       ERRMSG("Cannot create file!");
02410
02411     /* Write data... */
02412     FWRITE(&atm->np, int,
02413            1,
02414            out);
02415     FWRITE(atm->time, double,
02416              (size_t) atm->np,
02417            out);
02418     FWRITE(atm->p, double,
02419              (size_t) atm->np,
02420            out);
02421     FWRITE(atm->lon, double,
02422              (size_t) atm->np,
02423            out);
02424     FWRITE(atm->lat, double,
02425              (size_t) atm->np,
02426            out);
02427     for (iq = 0; iq < ctl->nq; iq++)
02428       FWRITE(atm->q[iq], double,
02429                (size_t) atm->np,
02430              out);
02431
02432     /* Close file... */
02433     fclose(out);
02434   }
02435
02436   /* Error... */
02437   else
02438     ERRMSG("Atmospheric data type not supported!");
02439 }
02440
02441 /*****************************************************************************/
02442
02443 void write_csi(
02444   const char *filename,
02445   ctl_t * ctl,
02446   atm_t * atm,
02447   double t) {
02448
02449   static FILE *in, *out;
02450
02451   static char line[LEN];
02452
02453   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02454     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02455
02456   static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02457
02458   /* Init... */
02459   if (t == ctl->t_start) {
02460
02461     /* Check quantity index for mass... */
02462     if (ctl->qnt_m < 0)
02463       ERRMSG("Need quantity mass!");
02464
02465     /* Open observation data file... */
02466     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02467     if (!(in = fopen(ctl->csi_obsfile, "r")))
02468       ERRMSG("Cannot open file!");
```

```
02469
02470     /* Create new file... */
02471     printf("Write CSI data: %s\n", filename);
02472     if (!(out = fopen(filename, "w")))
02473       ERRMSG("Cannot create file!");
02474
02475     /* Write header... */
02476     fprintf(out,
02477             "# $1 = time [s]\n"
02478             "# $2 = number of hits (cx)\n"
02479             "# $3 = number of misses (cy)\n"
02480             "# $4 = number of false alarms (cz)\n"
02481             "# $5 = number of observations (cx + cy)\n"
02482             "# $6 = number of forecasts (cx + cz)\n"
02483             "# $7 = bias (forecasts/observations) [%%]\n"
02484             "# $8 = probability of detection (POD) [%%]\n"
02485             "# $9 = false alarm rate (FAR) [%%]\n"
02486             "# $10 = critical success index (CSI) [%%]\n\n");
02487   }
02488
02489   /* Set time interval... */
02490   t0 = t - 0.5 * ctl->dt_mod;
02491   t1 = t + 0.5 * ctl->dt_mod;
02492
02493   /* Initialize grid cells... */
02494 #pragma omp parallel for default(shared) private(ix,iy,iz)
02495   for (ix = 0; ix < ctl->csi_nx; ix++)
02496     for (iy = 0; iy < ctl->csi_ny; iy++)
02497       for (iz = 0; iz < ctl->csi_nz; iz++)
02498         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02499
02500   /* Read observation data... */
02501   while (fgets(line, LEN, in)) {
02502
02503     /* Read data... */
02504     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02505         5)
02506       continue;
02507
02508     /* Check time... */
02509     if (rt < t0)
02510       continue;
02511     if (rt > t1)
02512       break;
02513
02514     /* Calculate indices... */
02515     ix = (int) ((rlon - ctl->csi_lon0)
02516                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02517     iy = (int) ((rlat - ctl->csi_lat0)
02518                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02519     iz = (int) ((rz - ctl->csi_z0)
02520                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02521
02522     /* Check indices... */
02523     if (ix < 0 || ix >= ctl->csi_nx ||
02524         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02525       continue;
02526
02527     /* Get mean observation index... */
02528     obsmean[ix][iy][iz] += robs;
02529     obscount[ix][iy][iz]++;
02530   }
02531
02532   /* Analyze model data... */
02533 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02534   for (ip = 0; ip < atm->np; ip++) {
02535
02536     /* Check time... */
02537     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02538       continue;
02539
02540     /* Get indices... */
02541     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02542                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02543     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02544                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02545     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02546                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02547
02548     /* Check indices... */
02549     if (ix < 0 || ix >= ctl->csi_nx ||
02550         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02551       continue;
02552
02553     /* Get total mass in grid cell... */
02554     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02555   }
```

```
02556
02557   /* Analyze all grid cells... */
02558 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02559   for (ix = 0; ix < ctl->csi_nx; ix++)
02560     for (iy = 0; iy < ctl->csi_ny; iy++)
02561       for (iz = 0; iz < ctl->csi_nz; iz++) {
02562
02563         /* Calculate mean observation index... */
02564         if (obscount[ix][iy][iz] > 0)
02565           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02566
02567         /* Calculate column density... */
02568         if (modmean[ix][iy][iz] > 0) {
02569           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02570           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02571           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02572           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02573             * cos(lat * M_PI / 180.);
02574           modmean[ix][iy][iz] /= (1e6 * area);
02575         }
02576
02577         /* Calculate CSI... */
02578         if (obscount[ix][iy][iz] > 0) {
02579           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02580               modmean[ix][iy][iz] >= ctl->csi_modmin)
02581             cx++;
02582           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02583                    modmean[ix][iy][iz] < ctl->csi_modmin)
02584             cy++;
02585           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02586                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02587             cz++;
02588         }
02589       }
02590
02591   /* Write output... */
02592   if (fmod(t, ctl->csi_dt_out) == 0) {
02593
02594     /* Write... */
02595     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02596             t, cx, cy, cz, cx + cy, cx + cz,
02597             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02598             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02599             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02600             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02601
02602     /* Set counters to zero... */
02603     cx = cy = cz = 0;
02604   }
02605
02606   /* Close file... */
02607   if (t == ctl->t_stop)
02608     fclose(out);
02609 }
02610
02611 /*****************************************************************************/
02612
02613 void write_ens(
02614   const char *filename,
02615   ctl_t * ctl,
02616   atm_t * atm,
02617   double t) {
02618
02619   static FILE *out;
02620
02621   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02622     t0, t1, x[NENS][3], xm[3];
02623
02624   static int ip, iq;
02625
02626   static size_t i, n;
02627
02628   /* Init... */
02629   if (t == ctl->t_start) {
02630
02631     /* Check quantities... */
02632     if (ctl->qnt_ens < 0)
02633       ERRMSG("Missing ensemble IDs!");
02634
02635     /* Create new file... */
02636     printf("Write ensemble data: %s\n", filename);
02637     if (!(out = fopen(filename, "w")))
02638       ERRMSG("Cannot create file!");
02639
02640     /* Write header... */
02641     fprintf(out,
02642             "# $1 = time [s]\n"
```

```
02643             "# $2 = altitude [km]\n"
02644             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02645     for (iq = 0; iq < ctl->nq; iq++)
02646       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02647               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02648     for (iq = 0; iq < ctl->nq; iq++)
02649       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02650               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02651     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02652   }
02653
02654   /* Set time interval... */
02655   t0 = t - 0.5 * ctl->dt_mod;
02656   t1 = t + 0.5 * ctl->dt_mod;
02657
02658   /* Init... */
02659   ens = GSL_NAN;
02660   n = 0;
02661
02662   /* Loop over air parcels... */
02663   for (ip = 0; ip < atm->np; ip++) {
02664
02665     /* Check time... */
02666     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02667       continue;
02668
02669     /* Check ensemble id... */
02670     if (atm->q[ctl->qnt_ens][ip] != ens) {
02671
02672       /* Write results... */
02673       if (n > 0) {
02674
02675         /* Get mean position... */
02676         xm[0] = xm[1] = xm[2] = 0;
02677         for (i = 0; i < n; i++) {
02678           xm[0] += x[i][0] / (double) n;
02679           xm[1] += x[i][1] / (double) n;
02680           xm[2] += x[i][2] / (double) n;
02681         }
02682         cart2geo(xm, &dummy, &lon, &lat);
02683         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02684                 lat);
02685
02686         /* Get quantity statistics... */
02687         for (iq = 0; iq < ctl->nq; iq++) {
02688           fprintf(out, " ");
02689           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02690         }
02691         for (iq = 0; iq < ctl->nq; iq++) {
02692           fprintf(out, " ");
02693           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02694         }
02695         fprintf(out, " %lu\n", n);
02696       }
02697
02698       /* Init new ensemble... */
02699       ens = atm->q[ctl->qnt_ens][ip];
02700       n = 0;
02701     }
02702
02703     /* Save data... */
02704     p[n] = atm->p[ip];
02705     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02706     for (iq = 0; iq < ctl->nq; iq++)
02707       q[iq][n] = atm->q[iq][ip];
02708     if ((++n) >= NENS)
02709       ERRMSG("Too many data points!");
02710   }
02711
02712   /* Write results... */
02713   if (n > 0) {
02714
02715     /* Get mean position... */
02716     xm[0] = xm[1] = xm[2] = 0;
02717     for (i = 0; i < n; i++) {
02718       xm[0] += x[i][0] / (double) n;
02719       xm[1] += x[i][1] / (double) n;
02720       xm[2] += x[i][2] / (double) n;
02721     }
02722     cart2geo(xm, &dummy, &lon, &lat);
02723     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02724
02725     /* Get quantity statistics... */
02726     for (iq = 0; iq < ctl->nq; iq++) {
02727       fprintf(out, " ");
02728       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02729     }
```

```
02730     for (iq = 0; iq < ctl->nq; iq++) {
02731       fprintf(out, " ");
02732       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02733     }
02734     fprintf(out, " %lu\n", n);
02735   }
02736
02737   /* Close file... */
02738   if (t == ctl->t_stop)
02739     fclose(out);
02740 }
02741
02742 /*****************************************************************************/
02743
02744 void write_grid(
02745   const char *filename,
02746   ctl_t * ctl,
02747   met_t * met0,
02748   met_t * met1,
02749   atm_t * atm,
02750   double t) {
02751
02752   FILE *in, *out;
02753
02754   char line[LEN];
02755
02756   static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02757     area, rho_air, press, temp, cd, vmr, t0, t1, r;
02758
02759   static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
02760
02761   /* Check dimensions... */
02762   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02763     ERRMSG("Grid dimensions too large!");
02764
02765   /* Set time interval for output... */
02766   t0 = t - 0.5 * ctl->dt_mod;
02767   t1 = t + 0.5 * ctl->dt_mod;
02768
02769   /* Set grid box size... */
02770   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02771   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
02772   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02773
02774   /* Initialize grid... */
02775 #pragma omp parallel for default(shared) private(ix,iy,iz)
02776   for (ix = 0; ix < ctl->grid_nx; ix++)
02777     for (iy = 0; iy < ctl->grid_ny; iy++)
02778       for (iz = 0; iz < ctl->grid_nz; iz++) {
02779         mass[ix][iy][iz] = 0;
02780         np[ix][iy][iz] = 0;
02781       }
02782
02783   /* Average data... */
02784 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02785   for (ip = 0; ip < atm->np; ip++)
02786     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02787
02788       /* Get index... */
02789       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02790       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02791       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02792
02793       /* Check indices... */
02794       if (ix < 0 || ix >= ctl->grid_nx ||
02795           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02796         continue;
02797
02798       /* Add mass... */
02799       if (ctl->qnt_m >= 0)
02800         mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02801       np[ix][iy][iz]++;
02802     }
02803
02804   /* Check if gnuplot output is requested... */
02805   if (ctl->grid_gpfile[0] != '-') {
02806
02807     /* Write info... */
02808     printf("Plot grid data: %s.png\n", filename);
02809
02810     /* Create gnuplot pipe... */
02811     if (!(out = popen("gnuplot", "w")))
02812       ERRMSG("Cannot create pipe to gnuplot!");
02813
02814     /* Set plot filename... */
02815     fprintf(out, "set out \"%s.png\"\n", filename);
02816
```

```
02817     /* Set time string... */
02818     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02819     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02820             year, mon, day, hour, min);
02821
02822     /* Dump gnuplot file to pipe... */
02823     if (!(in = fopen(ctl->grid_gpfile, "r")))
02824       ERRMSG("Cannot open file!");
02825     while (fgets(line, LEN, in))
02826       fprintf(out, "%s", line);
02827     fclose(in);
02828   }
02829
02830   else {
02831
02832     /* Write info... */
02833     printf("Write grid data: %s\n", filename);
02834
02835     /* Create file... */
02836     if (!(out = fopen(filename, "w")))
02837       ERRMSG("Cannot create file!");
02838   }
02839
02840   /* Write header... */
02841   fprintf(out,
02842           "# $1 = time [s]\n"
02843           "# $2 = altitude [km]\n"
02844           "# $3 = longitude [deg]\n"
02845           "# $4 = latitude [deg]\n"
02846           "# $5 = surface area [km^2]\n"
02847           "# $6 = layer width [km]\n"
02848           "# $7 = number of particles [1]\n"
02849           "# $8 = column density [kg/m^2]\n"
02850           "# $9 = volume mixing ratio [1]\n\n");
02851
02852   /* Write data... */
02853   for (ix = 0; ix < ctl->grid_nx; ix++) {
02854     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02855       fprintf(out, "\n");
02856     for (iy = 0; iy < ctl->grid_ny; iy++) {
02857       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02858         fprintf(out, "\n");
02859       for (iz = 0; iz < ctl->grid_nz; iz++)
02860         if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02861
02862           /* Set coordinates... */
02863           z = ctl->grid_z0 + dz * (iz + 0.5);
02864           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02865           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02866
02867           /* Get pressure and temperature... */
02868           press = P(z);
02869           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02870                           NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02871
02872           /* Calculate surface area... */
02873           area = dlat * dlon * SQR(RE * M_PI / 180.)
02874             * cos(lat * M_PI / 180.);
02875
02876           /* Calculate column density... */
02877           cd = mass[ix][iy][iz] / (1e6 * area);
02878
02879           /* Calculate volume mixing ratio... */
02880           rho_air = 100. * press / (RA * temp);
02881           vmr = MA / ctl->molmass * mass[ix][iy][iz]
02882             / (rho_air * 1e6 * area * 1e3 * dz);
02883
02884           /* Write output... */
02885           fprintf(out, "%.2f %g %g %g %g %d %g %g\n",
02886                   t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
02887         }
02888     }
02889   }
02890
02891   /* Close file... */
02892   fclose(out);
02893 }
02894
02895 /*****************************************************************************/
02896
02897 void write_prof(
02898   const char *filename,
02899   ctl_t * ctl,
02900   met_t * met0,
02901   met_t * met1,
02902   atm_t * atm,
02903   double t) {
```

```
02904
02905   static FILE *in, *out;
02906
02907   static char line[LEN];
02908
02909   static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
02910     rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
02911     rho_air, vmr, h2o, o3;
02912
02913   static int obscount[GX][GY], ip, ix, iy, iz, okay;
02914
02915   /* Init... */
02916   if (t == ctl->t_start) {
02917
02918     /* Check quantity index for mass... */
02919     if (ctl->qnt_m < 0)
02920       ERRMSG("Need quantity mass!");
02921
02922     /* Check dimensions... */
02923     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02924       ERRMSG("Grid dimensions too large!");
02925
02926     /* Open observation data file... */
02927     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02928     if (!(in = fopen(ctl->prof_obsfile, "r")))
02929       ERRMSG("Cannot open file!");
02930
02931     /* Create new output file... */
02932     printf("Write profile data: %s\n", filename);
02933     if (!(out = fopen(filename, "w")))
02934       ERRMSG("Cannot create file!");
02935
02936     /* Write header... */
02937     fprintf(out,
02938             "# $1 = time [s]\n"
02939             "# $2 = altitude [km]\n"
02940             "# $3 = longitude [deg]\n"
02941             "# $4 = latitude [deg]\n"
02942             "# $5 = pressure [hPa]\n"
02943             "# $6 = temperature [K]\n"
02944             "# $7 = volume mixing ratio [1]\n"
02945             "# $8 = H2O volume mixing ratio [1]\n"
02946             "# $9 = O3 volume mixing ratio [1]\n"
02947             "# $10 = observed BT index (mean) [K]\n"
02948             "# $11 = observed BT index (sigma) [K]\n");
02949
02950     /* Set grid box size... */
02951     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
02952     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02953     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02954   }
02955
02956   /* Set time interval... */
02957   t0 = t - 0.5 * ctl->dt_mod;
02958   t1 = t + 0.5 * ctl->dt_mod;
02959
02960   /* Initialize... */
02961 #pragma omp parallel for default(shared) private(ix,iy,iz)
02962   for (ix = 0; ix < ctl->prof_nx; ix++)
02963     for (iy = 0; iy < ctl->prof_ny; iy++) {
02964       obsmean[ix][iy] = 0;
02965       obsmean2[ix][iy] = 0;
02966       obscount[ix][iy] = 0;
02967       for (iz = 0; iz < ctl->prof_nz; iz++)
02968         mass[ix][iy][iz] = 0;
02969     }
02970
02971   /* Read observation data... */
02972   while (fgets(line, LEN, in)) {
02973
02974     /* Read data... */
02975     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02976         5)
02977       continue;
02978
02979     /* Check time... */
02980     if (rt < t0)
02981       continue;
02982     if (rt > t1)
02983       break;
02984
02985     /* Calculate indices... */
02986     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
02987     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02988
02989     /* Check indices... */
02990     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
```

```
02991       continue;
02992
02993     /* Get mean observation index... */
02994     obsmean[ix][iy] += robs;
02995     obsmean2[ix][iy] += SQR(robs);
02996     obscount[ix][iy]++;
02997   }
02998
02999   /* Analyze model data... */
03000 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03001   for (ip = 0; ip < atm->np; ip++) {
03002
03003     /* Check time... */
03004     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03005       continue;
03006
03007     /* Get indices... */
03008     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
03009     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
03010     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
03011
03012     /* Check indices... */
03013     if (ix < 0 || ix >= ctl->prof_nx ||
03014         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
03015       continue;
03016
03017     /* Get total mass in grid cell... */
03018     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03019   }
03020
03021   /* Extract profiles... */
03022   for (ix = 0; ix < ctl->prof_nx; ix++)
03023     for (iy = 0; iy < ctl->prof_ny; iy++)
03024       if (obscount[ix][iy] > 0) {
03025
03026         /* Check profile... */
03027         okay = 0;
03028         for (iz = 0; iz < ctl->prof_nz; iz++)
03029           if (mass[ix][iy][iz] > 0) {
03030             okay = 1;
03031             break;
03032           }
03033         if (!okay)
03034           continue;
03035
03036         /* Write output... */
03037         fprintf(out, "\n");
03038
03039         /* Loop over altitudes... */
03040         for (iz = 0; iz < ctl->prof_nz; iz++) {
03041
03042           /* Set coordinates... */
03043           z = ctl->prof_z0 + dz * (iz + 0.5);
03044           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
03045           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
03046
03047           /* Get pressure and temperature... */
03048           press = P(z);
03049           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
03050                           NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);
03051
03052           /* Calculate surface area... */
03053           area = dlat * dlon * SQR(M_PI * RE / 180.)
03054             * cos(lat * M_PI / 180.);
03055
03056           /* Calculate volume mixing ratio... */
03057           rho_air = 100. * press / (RA * temp);
03058           vmr = MA / ctl->molmass * mass[ix][iy][iz]
03059             / (rho_air * area * dz * 1e9);
03060
03061           /* Write output... */
03062           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n",
03063                   t, z, lon, lat, press, temp, vmr, h2o, o3,
03064                   obsmean[ix][iy] / obscount[ix][iy],
03065                   sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03066                        - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03067         }
03068       }
03069
03070   /* Close file... */
03071   if (t == ctl->t_stop)
03072     fclose(out);
03073 }
03074
03075 /*****************************************************************************/
03076
03077 void write_station(
```

```
03078   const char *filename,
03079   ctl_t * ctl,
03080   atm_t * atm,
03081   double t) {
03082
03083   static FILE *out;
03084
03085   static double rmax2, t0, t1, x0[3], x1[3];
03086
03087   static int ip, iq;
03088
03089   /* Init... */
03090   if (t == ctl->t_start) {
03091
03092     /* Write info... */
03093     printf("Write station data: %s\n", filename);
03094
03095     /* Create new file... */
03096     if (!(out = fopen(filename, "w")))
03097       ERRMSG("Cannot create file!");
03098
03099     /* Write header... */
03100     fprintf(out,
03101             "# $1 = time [s]\n"
03102             "# $2 = altitude [km]\n"
03103             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03104     for (iq = 0; iq < ctl->nq; iq++)
03105       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03106               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03107     fprintf(out, "\n");
03108
03109     /* Set geolocation and search radius... */
03110     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03111     rmax2 = SQR(ctl->stat_r);
03112   }
03113
03114   /* Set time interval for output... */
03115   t0 = t - 0.5 * ctl->dt_mod;
03116   t1 = t + 0.5 * ctl->dt_mod;
03117
03118   /* Loop over air parcels... */
03119   for (ip = 0; ip < atm->np; ip++) {
03120
03121     /* Check time... */
03122     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03123       continue;
03124
03125     /* Check station flag... */
03126     if (ctl->qnt_stat >= 0)
03127       if (atm->q[ctl->qnt_stat][ip])
03128         continue;
03129
03130     /* Get Cartesian coordinates... */
03131     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03132
03133     /* Check horizontal distance... */
03134     if (DIST2(x0, x1) > rmax2)
03135       continue;
03136
03137     /* Set station flag... */
03138     if (ctl->qnt_stat >= 0)
03139       atm->q[ctl->qnt_stat][ip] = 1;
03140
03141     /* Write data... */
03142     fprintf(out, "%.2f %g %g %g",
03143             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
03144     for (iq = 0; iq < ctl->nq; iq++) {
03145       fprintf(out, " ");
03146       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03147     }
03148     fprintf(out, "\n");
03149   }
03150
03151   /* Close file... */
03152   if (t == ctl->t_stop)
03153     fclose(out);
03154 }
```

5.21 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.
- void day2doy (int year, int mon, int day, int *doy)

  Get day of year from date.
- void doy2day (int year, int doy, int *mon, int *day)

  Get date from day of year.
- 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 get_met_replace (char *orig, char *search, char *repl)

  Replace template strings in filename.
- 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 *pv, 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 *pv, 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_irr (double *xx, int n, double x)

  Find array index for irregular grid.
- int locate_reg (double *xx, int n, double x)

  Find array index for regular grid.
- int 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.
- int 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_pv (met_t *met)
  Calculate potential vorticity.
- 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.

5.21.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

5.21.2 Function Documentation

5.21.2.1 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 }
```

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

Climatology of HNO3 volume mixing ratios.

Definition at line 45 of file libtrac.c.

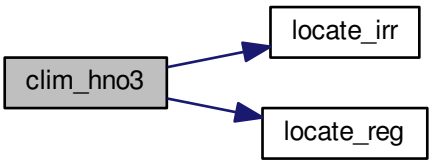
```
00048           {
00049
00050   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00051     9072000.00, 11664000.00, 14342400.00,
00052     16934400.00, 19612800.00, 22291200.00,
00053     24883200.00, 27561600.00, 30153600.00
00054   };
00055
00056   static double lats[18] = { -85, -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   isec = locate_irr(secs, 12, sec);
00292   ilat = locate_reg(lats, 18, lat);
00293   ip = locate_irr(ps, 10, p);
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:



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

Climatology of tropopause pressure.

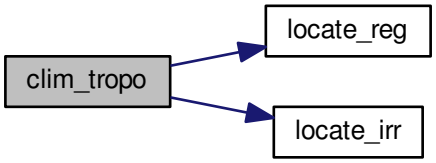
Definition at line 311 of file libtrac.c.

```
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;
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   ilat = locate_reg(lats, 73, lat);
00439   imon = locate_irr(doys, 12, doy);
```

```
00440
00441   /* Interpolate... */
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   return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
00447 }
```

Here is the call graph for this function:



5.21.2.4 void day2doy (int *year*, int *mon*, int *day*, int * *doy*)

Get day of year from date.

Definition at line 451 of file libtrac.c.

```
00455             {
00456
00457   int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00458   int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00459
00460   /* Get day of year... */
00461   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00462     *doy = d01[mon - 1] + day - 1;
00463   else
00464     *doy = d0[mon - 1] + day - 1;
00465 }
```

5.21.2.5 void doy2day (int *year*, int *doy*, int * *mon*, int * *day*)

Get date from day of year.

Definition at line 469 of file libtrac.c.

```
00473             {
00474
00475   int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00476   int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00477   int i;
00478
00479   /* Get month and day... */
00480   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00481     for (i = 11; i >= 0; i--)
00482       if (d01[i] <= doy)
00483         break;
00484     *mon = i + 1;
00485     *day = doy - d01[i] + 1;
00486   } else {
00487     for (i = 11; i >= 0; i--)
00488       if (d0[i] <= doy)
00489         break;
00490     *mon = i + 1;
00491     *day = doy - d0[i] + 1;
00492   }
00493 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 497 of file libtrac.c.

```
00501             {
00502
00503   double radius;
00504
00505   radius = z + RE;
00506   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00507   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00508   x[2] = radius * sin(lat / 180 * M_PI);
00509 }
```

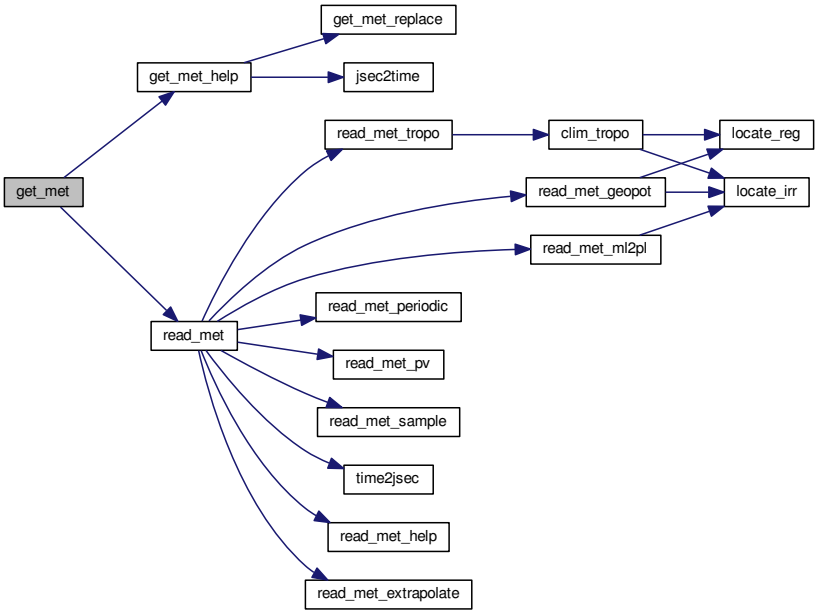
5.21.2.7 void get_met (ctl_t * *ctl*, char * *metbase*, double *t*, met_t ** *met0*, met_t ** *met1*)

Get meteorological data for given timestep.

Definition at line 513 of file libtrac.c.

```
00518                 {
00519
00520   static int init, ip, ix, iy;
00521
00522   met_t *mets;
00523
00524   char filename[LEN];
00525
00526   /* Init... */
00527   if (t == ctl->t_start || !init) {
00528     init = 1;
00529
00530     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00531     if (!read_met(ctl, filename, *met0))
00532       ERRMSG("Cannot open file!");
00533
00534     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00535     if (!read_met(ctl, filename, *met1))
00536       ERRMSG("Cannot open file!");
00537   }
00538
00539   /* Read new data for forward trajectories... */
00540   if (t > (*met1)->time && ctl->direction == 1) {
00541     mets = *met1;
00542     *met1 = *met0;
00543     *met0 = mets;
00544     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00545     if (!read_met(ctl, filename, *met1))
00546       ERRMSG("Cannot open file!");
00547   }
00548
00549   /* Read new data for backward trajectories... */
00550   if (t < (*met0)->time && ctl->direction == -1) {
00551     mets = *met1;
00552     *met1 = *met0;
00553     *met0 = mets;
00554     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00555     if (!read_met(ctl, filename, *met0))
00556       ERRMSG("Cannot open file!");
00557   }
00558
00559   /* Check that grids are consistent... */
00560   if ((*met0)->nx != (*met1)->nx
00561       || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
00562     ERRMSG("Meteo grid dimensions do not match!");
00563   for (ix = 0; ix < (*met0)->nx; ix++)
00564     if ((*met0)->lon[ix] != (*met1)->lon[ix])
00565       ERRMSG("Meteo grid longitudes do not match!");
00566   for (iy = 0; iy < (*met0)->ny; iy++)
00567     if ((*met0)->lat[iy] != (*met1)->lat[iy])
00568       ERRMSG("Meteo grid latitudes do not match!");
00569   for (ip = 0; ip < (*met0)->np; ip++)
00570     if ((*met0)->p[ip] != (*met1)->p[ip])
00571       ERRMSG("Meteo grid pressure levels do not match!");
00572 }
```

Here is the call graph for this function:



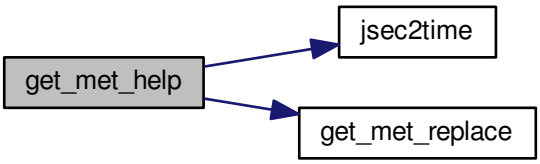
5.21.2.8 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

Get meteorological data for timestep.

Definition at line 576 of file libtrac.c.

```
00581                   {
00582
00583   char repl[LEN];
00584
00585   double t6, r;
00586
00587   int year, mon, day, hour, min, sec;
00588
00589   /* Round time to fixed intervals... */
00590   if (direct == -1)
00591     t6 = floor(t / dt_met) * dt_met;
00592   else
00593     t6 = ceil(t / dt_met) * dt_met;
00594
00595   /* Decode time... */
00596   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00597
00598   /* Set filename... */
00599   sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
00600   sprintf(repl, "%d", year);
00601   get_met_replace(filename, "YYYY", repl);
00602   sprintf(repl, "%02d", mon);
00603   get_met_replace(filename, "MM", repl);
00604   sprintf(repl, "%02d", day);
00605   get_met_replace(filename, "DD", repl);
00606   sprintf(repl, "%02d", hour);
00607   get_met_replace(filename, "HH", repl);
00608 }
```

Here is the call graph for this function:



5.21.2.9 void get_met_replace (char * *orig*, char * *search*, char * *repl*)

Replace template strings in filename.

Definition at line 612 of file libtrac.c.

```
00615               {
00616
00617   char buffer[LEN], *ch;
00618
00619   int i;
00620
00621   /* Iterate... */
00622   for (i = 0; i < 3; i++) {
00623
00624     /* Replace substring... */
00625     if (!(ch = strstr(orig, search)))
00626       return;
00627     strncpy(buffer, orig, (size_t) (ch - orig));
00628     buffer[ch - orig] = 0;
00629     sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
00630     orig[0] = 0;
00631     strcpy(orig, buffer);
00632   }
00633 }
```

5.21.2.10 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 637 of file libtrac.c.

```
00643                {
00644
00645   double aux00, aux01, aux10, aux11;
00646
00647   /* Set variables... */
00648   aux00 = array[ix][iy];
00649   aux01 = array[ix][iy + 1];
00650   aux10 = array[ix + 1][iy];
00651   aux11 = array[ix + 1][iy + 1];
00652
00653   /* Interpolate horizontally... */
00654   aux00 = wy * (aux00 - aux01) + aux01;
00655   aux11 = wy * (aux10 - aux11) + aux11;
00656   *var = wx * (aux00 - aux11) + aux11;
00657 }
```

5.21.2.11 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 661 of file libtrac.c.

```
00669                {
00670
00671   double aux00, aux01, aux10, aux11;
00672
00673   /* Interpolate vertically... */
00674   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00675     + array[ix][iy][ip + 1];
00676   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00677     + array[ix][iy + 1][ip + 1];
00678   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00679     + array[ix + 1][iy][ip + 1];
00680   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00681     + array[ix + 1][iy + 1][ip + 1];
00682
00683   /* Interpolate horizontally... */
00684   aux00 = wy * (aux00 - aux01) + aux01;
00685   aux11 = wy * (aux10 - aux11) + aux11;
00686   *var = wx * (aux00 - aux11) + aux11;
00687 }
```

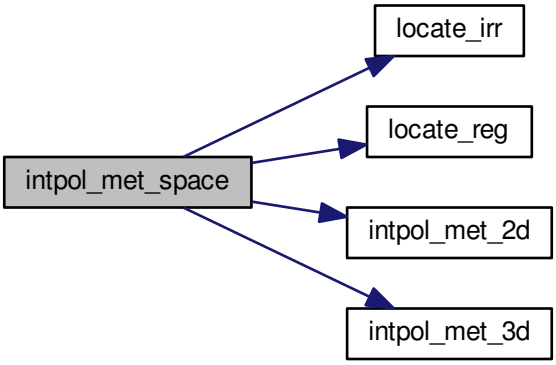
5.21.2.12 void intpol_met_space (met_t * *met*, double *p*, double *lon*, double *lat*, double * *ps*, double * *pt*, double * *z*, double * *t*, double * *u*, double * *v*, double * *w*, double * *pv*, double * *h2o*, double * *o3*)

Spatial interpolation of meteorological data.

Definition at line 691 of file libtrac.c.

```
00705                {
00706
00707   double wp, wx, wy;
00708
00709   int ip, ix, iy;
00710
00711   /* Check longitude... */
00712   if (met->lon[met->nx - 1] > 180 && lon < 0)
00713     lon += 360;
00714
00715   /* Get indices... */
00716   ip = locate_irr(met->p, met->np, p);
00717   ix = locate_reg(met->lon, met->nx, lon);
00718   iy = locate_reg(met->lat, met->ny, lat);
00719
00720   /* Get weights... */
00721   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00722   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00723   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00724
00725   /* Interpolate... */
00726   if (ps != NULL)
00727     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00728   if (pt != NULL)
00729     intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
00730   if (z != NULL)
00731     intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
00732   if (t != NULL)
00733     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00734   if (u != NULL)
00735     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00736   if (v != NULL)
00737     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00738   if (w != NULL)
00739     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00740   if (pv != NULL)
00741     intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
00742   if (h2o != NULL)
00743     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00744   if (o3 != NULL)
00745     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00746 }
```

Here is the call graph for this function:



5.21.2.13 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 * *pv*, double * *h2o*, double * *o3*)

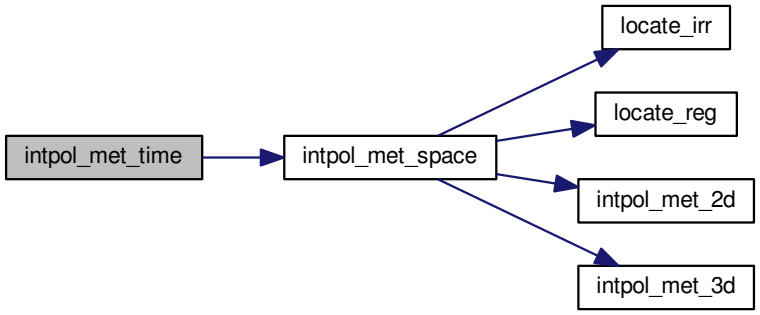
Temporal interpolation of meteorological data.

Definition at line 750 of file libtrac.c.

```
00766              {
00767
00768   double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
00769     v0, v1, w0, w1, wt, z0, z1;
00770
00771   /* Spatial interpolation... */
00772   intpol_met_space(met0, p, lon, lat,
00773                    ps == NULL ? NULL : &ps0,
00774                    pt == NULL ? NULL : &pt0,
00775                    z == NULL ? NULL : &z0,
00776                    t == NULL ? NULL : &t0,
00777                    u == NULL ? NULL : &u0,
00778                    v == NULL ? NULL : &v0,
00779                    w == NULL ? NULL : &w0,
00780                    pv == NULL ? NULL : &pv0,
00781                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00782   intpol_met_space(met1, p, lon, lat,
00783                    ps == NULL ? NULL : &ps1,
00784                    pt == NULL ? NULL : &pt1,
00785                    z == NULL ? NULL : &z1,
00786                    t == NULL ? NULL : &t1,
00787                    u == NULL ? NULL : &u1,
00788                    v == NULL ? NULL : &v1,
00789                    w == NULL ? NULL : &w1,
00790                    pv == NULL ? NULL : &pv1,
00791                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00792
00793   /* Get weighting factor... */
00794   wt = (met1->time - ts) / (met1->time - met0->time);
00795
00796   /* Interpolate... */
00797   if (ps != NULL)
00798     *ps = wt * (ps0 - ps1) + ps1;
00799   if (pt != NULL)
00800     *pt = wt * (pt0 - pt1) + pt1;
00801   if (z != NULL)
00802     *z = wt * (z0 - z1) + z1;
00803   if (t != NULL)
00804     *t = wt * (t0 - t1) + t1;
00805   if (u != NULL)
00806     *u = wt * (u0 - u1) + u1;
```

```
00807   if (v != NULL)
00808     *v = wt * (v0 - v1) + v1;
00809   if (w != NULL)
00810     *w = wt * (w0 - w1) + w1;
00811   if (pv != NULL)
00812     *pv = wt * (pv0 - pv1) + pv1;
00813   if (h2o != NULL)
00814     *h2o = wt * (h2o0 - h2o1) + h2o1;
00815   if (o3 != NULL)
00816     *o3 = wt * (o30 - o31) + o31;
00817 }
```

Here is the call graph for this function:



5.21.2.14 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 821 of file libtrac.c.

```
00829                   {
00830
00831   struct tm t0, *t1;
00832
00833   time_t jsec0;
00834
00835   t0.tm_year = 100;
00836   t0.tm_mon = 0;
00837   t0.tm_mday = 1;
00838   t0.tm_hour = 0;
00839   t0.tm_min = 0;
00840   t0.tm_sec = 0;
00841
00842   jsec0 = (time_t) jsec + timegm(&t0);
00843   t1 = gmtime(&jsec0);
00844
00845   *year = t1->tm_year + 1900;
00846   *mon = t1->tm_mon + 1;
00847   *day = t1->tm_mday;
00848   *hour = t1->tm_hour;
00849   *min = t1->tm_min;
00850   *sec = t1->tm_sec;
00851   *remain = jsec - floor(jsec);
00852 }
```

5.21.2.15 int locate_irr (double * *xx*, int *n*, double *x*)

Find array index for irregular grid.

Definition at line 856 of file libtrac.c.

```
00859             {
00860
00861   int i, ilo, ihi;
00862
00863   ilo = 0;
00864   ihi = n - 1;
00865   i = (ihi + ilo) >> 1;
00866
00867   if (xx[i] < xx[i + 1])
00868     while (ihi > ilo + 1) {
00869       i = (ihi + ilo) >> 1;
00870       if (xx[i] > x)
00871         ihi = i;
00872       else
00873         ilo = i;
00874   } else
00875     while (ihi > ilo + 1) {
00876       i = (ihi + ilo) >> 1;
00877       if (xx[i] <= x)
00878         ihi = i;
00879       else
00880         ilo = i;
00881     }
00882
00883   return ilo;
00884 }
```

5.21.2.16 int locate_reg (double * *xx*, int *n*, double *x*)

Find array index for regular grid.

Definition at line 888 of file libtrac.c.

```
00891             {
00892
00893   int i;
00894
00895   /* Calculate index... */
00896   i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
00897
00898   /* Check range... */
00899   if (i < 0)
00900     i = 0;
00901   else if (i >= n - 2)
00902     i = n - 2;
00903
00904   return i;
00905 }
```

5.21.2.17 int read_atm (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Read atmospheric data.

Definition at line 909 of file libtrac.c.

```
00912               {
00913
00914   FILE *in;
00915
00916   char line[LEN], *tok;
00917
00918   double t0;
00919
00920   int dimid, ip, iq, ncid, varid;
00921
00922   size_t nparts;
00923
00924   /* Init... */
00925   atm->np = 0;
00926
00927   /* Write info... */
00928   printf("Read atmospheric data: %s\n", filename);
00929
00930   /* Read ASCII data... */
00931   if (ctl->atm_type == 0) {
00932
00933     /* Open file... */
00934     if (!(in = fopen(filename, "r")))
00935       return 0;
00936
00937     /* Read line... */
00938     while (fgets(line, LEN, in)) {
00939
00940       /* Read data... */
00941       TOK(line, tok, "%lg", atm->time[atm->np]);
00942       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00943       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00944       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00945       for (iq = 0; iq < ctl->nq; iq++)
00946         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00947
00948       /* Convert altitude to pressure... */
00949       atm->p[atm->np] = P(atm->p[atm->np]);
00950
00951       /* Increment data point counter... */
00952       if ((++atm->np) > NP)
00953         ERRMSG("Too many data points!");
00954     }
00955
00956     /* Close file... */
00957     fclose(in);
00958   }
00959
00960   /* Read binary data... */
00961   else if (ctl->atm_type == 1) {
00962
00963     /* Open file... */
00964     if (!(in = fopen(filename, "r")))
00965       return 0;
00966
00967     /* Read data... */
00968     FREAD(&atm->np, int, 1, in);
00969     FREAD(atm->time, double,
00970           (size_t) atm->np,
00971           in);
00972     FREAD(atm->p, double,
00973           (size_t) atm->np,
00974           in);
00975     FREAD(atm->lon, double,
00976           (size_t) atm->np,
00977           in);
00978     FREAD(atm->lat, double,
00979           (size_t) atm->np,
00980           in);
00981     for (iq = 0; iq < ctl->nq; iq++)
00982       FREAD(atm->q[iq], double,
00983             (size_t) atm->np,
00984             in);
00985
00986     /* Close file... */
00987     fclose(in);
00988   }
00989
00990   /* Read netCDF data... */
00991   else if (ctl->atm_type == 2) {
00992
00993     /* Open file... */
00994     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
00995       return 0;
00996
00997     /* Get dimensions... */
00998     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
```

```
00999     NC(nc_inq_dimlen(ncid, dimid, &nparts));
01000     atm->np = (int) nparts;
01001     if (atm->np > NP)
01002       ERRMSG("Too many particles!");
01003
01004     /* Get time... */
01005     NC(nc_inq_varid(ncid, "time", &varid));
01006     NC(nc_get_var_double(ncid, varid, &t0));
01007     for (ip = 0; ip < atm->np; ip++)
01008       atm->time[ip] = t0;
01009
01010     /* Read geolocations... */
01011     NC(nc_inq_varid(ncid, "PRESS", &varid));
01012     NC(nc_get_var_double(ncid, varid, atm->p));
01013     NC(nc_inq_varid(ncid, "LON", &varid));
01014     NC(nc_get_var_double(ncid, varid, atm->lon));
01015     NC(nc_inq_varid(ncid, "LAT", &varid));
01016     NC(nc_get_var_double(ncid, varid, atm->lat));
01017
01018     /* Read variables... */
01019     if (ctl->qnt_p >= 0)
01020       if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
01021         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
01022     if (ctl->qnt_t >= 0)
01023       if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
01024         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
01025     if (ctl->qnt_u >= 0)
01026       if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
01027         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
01028     if (ctl->qnt_v >= 0)
01029       if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
01030         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
01031     if (ctl->qnt_w >= 0)
01032       if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
01033         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
01034     if (ctl->qnt_h2o >= 0)
01035       if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
01036         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
01037     if (ctl->qnt_o3 >= 0)
01038       if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
01039         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
01040     if (ctl->qnt_theta >= 0)
01041       if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
01042         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
01043     if (ctl->qnt_pv >= 0)
01044       if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
01045         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
01046
01047     /* Check data... */
01048     for (ip = 0; ip < atm->np; ip++)
01049       if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
01050           || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
01051           || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
01052           || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
01053           || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
01054         atm->time[ip] = GSL_NAN;
01055         atm->p[ip] = GSL_NAN;
01056         atm->lon[ip] = GSL_NAN;
01057         atm->lat[ip] = GSL_NAN;
01058         for (iq = 0; iq < ctl->nq; iq++)
01059           atm->q[iq][ip] = GSL_NAN;
01060       } else {
01061         if (ctl->qnt_h2o >= 0)
01062           atm->q[ctl->qnt_h2o][ip] *= 1.608;
01063         if (ctl->qnt_pv >= 0)
01064           atm->q[ctl->qnt_pv][ip] *= 1e6;
01065         if (atm->lon[ip] > 180)
01066           atm->lon[ip] -= 360;
01067       }
01068
01069     /* Close file... */
01070     NC(nc_close(ncid));
01071   }
01072
01073   /* Error... */
01074   else
01075     ERRMSG("Atmospheric data type not supported!");
01076
01077   /* Check number of points... */
01078   if (atm->np < 1)
01079     ERRMSG("Can not read any data!");
01080
01081   /* Return success... */
01082   return 1;
01083 }
```

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

Read control parameters.

Definition at line 1087 of file libtrac.c.

```
01091               {
01092
01093   int ip, iq;
01094
01095   /* Write info... */
01096   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
01097          "(executable: %s | compiled: %s, %s)\n\n",
01098          argv[0], __DATE__, __TIME__);
01099
01100   /* Initialize quantity indices... */
01101   ctl->qnt_ens = -1;
01102   ctl->qnt_m = -1;
01103   ctl->qnt_r = -1;
01104   ctl->qnt_rho = -1;
01105   ctl->qnt_ps = -1;
01106   ctl->qnt_pt = -1;
01107   ctl->qnt_z = -1;
01108   ctl->qnt_p = -1;
01109   ctl->qnt_t = -1;
01110   ctl->qnt_u = -1;
01111   ctl->qnt_v = -1;
01112   ctl->qnt_w = -1;
01113   ctl->qnt_h2o = -1;
01114   ctl->qnt_o3 = -1;
01115   ctl->qnt_theta = -1;
01116   ctl->qnt_vh = -1;
01117   ctl->qnt_vz = -1;
01118   ctl->qnt_pv = -1;
01119   ctl->qnt_tice = -1;
01120   ctl->qnt_tsts = -1;
01121   ctl->qnt_tnat = -1;
01122   ctl->qnt_stat = -1;
01123
01124   /* Read quantities... */
01125   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
01126   if (ctl->nq > NQ)
01127     ERRMSG("Too many quantities!");
01128   for (iq = 0; iq < ctl->nq; iq++) {
01129
01130     /* Read quantity name and format... */
01131     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
01132     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
01133              ctl->qnt_format[iq]);
01134
01135     /* Try to identify quantity... */
01136     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
01137       ctl->qnt_ens = iq;
01138       sprintf(ctl->qnt_unit[iq], "-");
01139     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
01140       ctl->qnt_m = iq;
01141       sprintf(ctl->qnt_unit[iq], "kg");
01142     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
01143       ctl->qnt_r = iq;
01144       sprintf(ctl->qnt_unit[iq], "m");
01145     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
01146       ctl->qnt_rho = iq;
01147       sprintf(ctl->qnt_unit[iq], "kg/m^3");
01148     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
01149       ctl->qnt_ps = iq;
01150       sprintf(ctl->qnt_unit[iq], "hPa");
01151     } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
01152       ctl->qnt_pt = iq;
01153       sprintf(ctl->qnt_unit[iq], "hPa");
01154     } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
01155       ctl->qnt_z = iq;
01156       sprintf(ctl->qnt_unit[iq], "km");
01157     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
01158       ctl->qnt_p = iq;
01159       sprintf(ctl->qnt_unit[iq], "hPa");
01160     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
01161       ctl->qnt_t = iq;
01162       sprintf(ctl->qnt_unit[iq], "K");
01163     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
01164       ctl->qnt_u = iq;
01165       sprintf(ctl->qnt_unit[iq], "m/s");
01166     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
01167       ctl->qnt_v = iq;
```

```
01168       sprintf(ctl->qnt_unit[iq], "m/s");
01169     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
01170       ctl->qnt_w = iq;
01171       sprintf(ctl->qnt_unit[iq], "hPa/s");
01172     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
01173       ctl->qnt_h2o = iq;
01174       sprintf(ctl->qnt_unit[iq], "1");
01175     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
01176       ctl->qnt_o3 = iq;
01177       sprintf(ctl->qnt_unit[iq], "1");
01178     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
01179       ctl->qnt_theta = iq;
01180       sprintf(ctl->qnt_unit[iq], "K");
01181     } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
01182       ctl->qnt_vh = iq;
01183       sprintf(ctl->qnt_unit[iq], "m/s");
01184     } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
01185       ctl->qnt_vz = iq;
01186       sprintf(ctl->qnt_unit[iq], "m/s");
01187     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
01188       ctl->qnt_pv = iq;
01189       sprintf(ctl->qnt_unit[iq], "PVU");
01190     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
01191       ctl->qnt_tice = iq;
01192       sprintf(ctl->qnt_unit[iq], "K");
01193     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
01194       ctl->qnt_tsts = iq;
01195       sprintf(ctl->qnt_unit[iq], "K");
01196     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
01197       ctl->qnt_tnat = iq;
01198       sprintf(ctl->qnt_unit[iq], "K");
01199     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
01200       ctl->qnt_stat = iq;
01201       sprintf(ctl->qnt_unit[iq], "-");
01202     } else
01203       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
01204   }
01205
01206   /* Time steps of simulation... */
01207   ctl->direction =
01208     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
01209   if (ctl->direction != -1 && ctl->direction != 1)
01210     ERRMSG("Set DIRECTION to -1 or 1!");
01211   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
01212   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
01213
01214   /* Meteorological data... */
01215   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
01216   ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
01217   ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
01218   ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
01219   ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
01220   ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
01221   ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
01222   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
01223   if (ctl->met_np > EP)
01224     ERRMSG("Too many levels!");
01225   for (ip = 0; ip < ctl->met_np; ip++)
01226     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
01227   ctl->met_tropo
01228     = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
01229   scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
01230   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
01231   ctl->met_dt_out =
01232     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
01233
01234   /* Isosurface parameters... */
01235   ctl->isosurf
01236     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
01237   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
01238
01239   /* Diffusion parameters... */
01240   ctl->turb_dx_trop
01241     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
01242   ctl->turb_dx_strat
01243     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
01244   ctl->turb_dz_trop
01245     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
01246   ctl->turb_dz_strat
01247     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
01248   ctl->turb_mesox =
01249     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
01250   ctl->turb_mesoz =
01251     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
01252
01253   /* Mass and life time... */
01254   ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
```

```
01255   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
01256   ctl->tdec_strat =
01257     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
01258
01259   /* PSC analysis... */
01260   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
01261   ctl->psc_hno3 =
01262     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
01263
01264   /* Output of atmospheric data... */
01265   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
01266   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
01267   ctl->atm_dt_out =
01268     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
01269   ctl->atm_filter =
01270     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
01271   ctl->atm_type =
01272     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
01273
01274   /* Output of CSI data... */
01275   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
01276   ctl->csi_dt_out =
01277     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
01278   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
      csi_obsfile);
01279   ctl->csi_obsmin =
01280     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
01281   ctl->csi_modmin =
01282     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
01283   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
01284   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
01285   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
01286   ctl->csi_lon0 =
01287     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
01288   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
01289   ctl->csi_nx =
01290     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
01291   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
01292   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
01293   ctl->csi_ny =
01294     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
01295
01296   /* Output of ensemble data... */
01297   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
01298
01299   /* Output of grid data... */
01300   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
01301            ctl->grid_basename);
01302   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
01303   ctl->grid_dt_out =
01304     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
01305   ctl->grid_sparse =
01306     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
01307   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
01308   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
01309   ctl->grid_nz =
01310     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
01311   ctl->grid_lon0 =
01312     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
01313   ctl->grid_lon1 =
01314     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
01315   ctl->grid_nx =
01316     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
01317   ctl->grid_lat0 =
01318     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
01319   ctl->grid_lat1 =
01320     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
01321   ctl->grid_ny =
01322     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
01323
01324   /* Output of profile data... */
01325   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
01326            ctl->prof_basename);
01327   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
01328   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
01329   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
01330   ctl->prof_nz =
01331     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
01332   ctl->prof_lon0 =
01333     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
01334   ctl->prof_lon1 =
01335     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
```

```
01336   ctl->prof_nx =
01337     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
01338   ctl->prof_lat0 =
01339     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
01340   ctl->prof_lat1 =
01341     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
01342   ctl->prof_ny =
01343     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
01344
01345   /* Output of station data... */
01346   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
01347            ctl->stat_basename);
01348   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
01349   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
01350   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
01351 }
```

Here is the call graph for this function:

5.21.2.19 int read_met (ctl_t * *ctl*, char * *filename*, met_t * *met*)

Read meteorological data file.

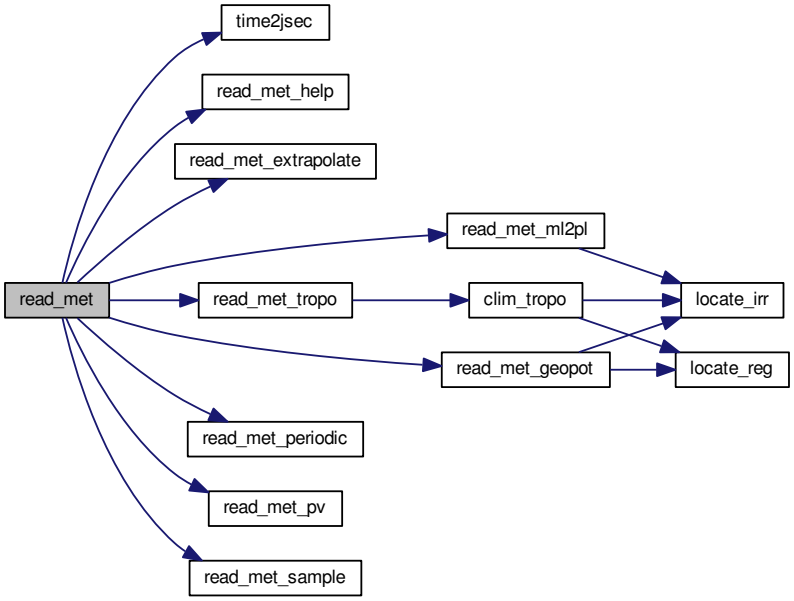
Definition at line 1355 of file libtrac.c.

```
01358               {
01359
01360   char cmd[2 * LEN], levname[LEN], tstr[10];
01361
01362   float help[EX * EY];
01363
01364   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
01365
01366   size_t np, nx, ny;
01367
01368   /* Write info... */
01369   printf("Read meteorological data: %s\n", filename);
01370
01371   /* Open netCDF file... */
01372   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
01373
01374     /* Try to stage meteo file... */
01375     if (ctl->met_stage[0] != '-') {
01376       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
01377               year, mon, day, hour, filename);
01378       if (system(cmd) != 0)
01379         ERRMSG("Error while staging meteo data!");
01380     }
01381
01382     /* Try to open again... */
01383     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
01384       return 0;
01385   }
01386
01387   /* Get time from filename... */
01388   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
01389   year = atoi(tstr);
01390   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
01391   mon = atoi(tstr);
01392   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
01393   day = atoi(tstr);
01394   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
```

```
01395   hour = atoi(tstr);
01396   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
01397
01398   /* Get dimensions... */
01399   NC(nc_inq_dimid(ncid, "lon", &dimid));
01400   NC(nc_inq_dimlen(ncid, dimid, &nx));
01401   if (nx < 2 || nx > EX)
01402     ERRMSG("Number of longitudes out of range!");
01403
01404   NC(nc_inq_dimid(ncid, "lat", &dimid));
01405   NC(nc_inq_dimlen(ncid, dimid, &ny));
01406   if (ny < 2 || ny > EY)
01407     ERRMSG("Number of latitudes out of range!");
01408
01409   sprintf(levname, "lev");
01410   NC(nc_inq_dimid(ncid, levname, &dimid));
01411   NC(nc_inq_dimlen(ncid, dimid, &np));
01412   if (np == 1) {
01413     sprintf(levname, "lev_2");
01414     NC(nc_inq_dimid(ncid, levname, &dimid));
01415     NC(nc_inq_dimlen(ncid, dimid, &np));
01416   }
01417   if (np < 2 || np > EP)
01418     ERRMSG("Number of levels out of range!");
01419
01420   /* Store dimensions... */
01421   met->np = (int) np;
01422   met->nx = (int) nx;
01423   met->ny = (int) ny;
01424
01425   /* Get horizontal grid... */
01426   NC(nc_inq_varid(ncid, "lon", &varid));
01427   NC(nc_get_var_double(ncid, varid, met->lon));
01428   NC(nc_inq_varid(ncid, "lat", &varid));
01429   NC(nc_get_var_double(ncid, varid, met->lat));
01430
01431   /* Read meteorological data... */
01432   read_met_help(ncid, "t", "T", met, met->t, 1.0);
01433   read_met_help(ncid, "u", "U", met, met->u, 1.0);
01434   read_met_help(ncid, "v", "V", met, met->v, 1.0);
01435   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
01436   read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
01437   read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));
01438
01439   /* Meteo data on pressure levels... */
01440   if (ctl->met_np <= 0) {
01441
01442     /* Read pressure levels from file... */
01443     NC(nc_inq_varid(ncid, levname, &varid));
01444     NC(nc_get_var_double(ncid, varid, met->p));
01445     for (ip = 0; ip < met->np; ip++)
01446       met->p[ip] /= 100.;
01447
01448     /* Extrapolate data for lower boundary... */
01449     read_met_extrapolate(met);
01450   }
01451
01452   /* Meteo data on model levels... */
01453   else {
01454
01455     /* Read pressure data from file... */
01456     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
01457
01458     /* Interpolate from model levels to pressure levels... */
01459     read_met_ml2pl(ctl, met, met->t);
01460     read_met_ml2pl(ctl, met, met->u);
01461     read_met_ml2pl(ctl, met, met->v);
01462     read_met_ml2pl(ctl, met, met->w);
01463     read_met_ml2pl(ctl, met, met->h2o);
01464     read_met_ml2pl(ctl, met, met->o3);
01465
01466     /* Set pressure levels... */
01467     met->np = ctl->met_np;
01468     for (ip = 0; ip < met->np; ip++)
01469       met->p[ip] = ctl->met_p[ip];
01470   }
01471
01472   /* Check ordering of pressure levels... */
01473   for (ip = 1; ip < met->np; ip++)
01474     if (met->p[ip - 1] < met->p[ip])
01475       ERRMSG("Pressure levels must be descending!");
01476
01477   /* Read surface pressure... */
01478   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
01479       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
01480     NC(nc_get_var_float(ncid, varid, help));
01481     for (iy = 0; iy < met->ny; iy++)
```

```
01482       for (ix = 0; ix < met->nx; ix++)
01483         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
01484     } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
01485                || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
01486       NC(nc_get_var_float(ncid, varid, help));
01487       for (iy = 0; iy < met->ny; iy++)
01488         for (ix = 0; ix < met->nx; ix++)
01489           met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
01490     } else
01491       for (ix = 0; ix < met->nx; ix++)
01492         for (iy = 0; iy < met->ny; iy++)
01493           met->ps[ix][iy] = met->p[0];
01494
01495     /* Create periodic boundary conditions... */
01496     read_met_periodic(met);
01497
01498     /* Calculate geopotential heights... */
01499     read_met_geopot(ctl, met);
01500
01501     /* Calculate potential vorticity... */
01502     read_met_pv(met);
01503
01504     /* Calculate tropopause pressure... */
01505     read_met_tropo(ctl, met);
01506
01507     /* Downsampling... */
01508     read_met_sample(ctl, met);
01509
01510     /* Close file... */
01511     NC(nc_close(ncid));
01512
01513     /* Return success... */
01514     return 1;
01515   }
```

Here is the call graph for this function:



5.21.2.20 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 1519 of file libtrac.c.

```
01520               {
01521
01522   int ip, ip0, ix, iy;
01523
01524   /* Loop over columns... */
01525 #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
01526   for (ix = 0; ix < met->nx; ix++)
01527     for (iy = 0; iy < met->ny; iy++) {
01528
01529       /* Find lowest valid data point... */
01530       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
01531         if (!gsl_finite(met->t[ix][iy][ip0])
01532             || !gsl_finite(met->u[ix][iy][ip0])
01533             || !gsl_finite(met->v[ix][iy][ip0])
01534             || !gsl_finite(met->w[ix][iy][ip0]))
01535           break;
01536
01537       /* Extrapolate... */
01538       for (ip = ip0; ip >= 0; ip--) {
01539         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
01540         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
01541         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
01542         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
01543         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
01544         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
01545       }
01546     }
01547 }
```

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

Calculate geopotential heights.

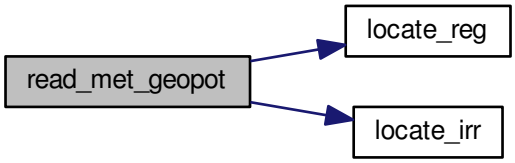
Definition at line 1551 of file libtrac.c.

```
01553               {
01554
01555   static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];
01556
01557   static int init, topo_nx = -1, topo_ny;
01558
01559   FILE *in;
01560
01561   char line[LEN];
01562
01563   double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;
01564
01565   float help[EX][EY];
01566
01567   int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;
01568
01569   /* Initialize geopotential heights... */
01570 #pragma omp parallel for default(shared) private(ix,iy,ip)
01571   for (ix = 0; ix < met->nx; ix++)
01572     for (iy = 0; iy < met->ny; iy++)
01573       for (ip = 0; ip < met->np; ip++)
01574         met->z[ix][iy][ip] = GSL_NAN;
01575
01576   /* Check filename... */
01577   if (ctl->met_geopot[0] == '-')
01578     return;
01579
01580   /* Read surface geopotential... */
01581   if (!init) {
01582     init = 1;
01583
01584     /* Write info... */
01585     printf("Read surface geopotential: %s\n", ctl->met_geopot);
01586
01587     /* Open file... */
01588     if (!(in = fopen(ctl->met_geopot, "r")))
01589       ERRMSG("Cannot open file!");
01590
01591     /* Read data... */
01592     while (fgets(line, LEN, in))
01593       if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
01594         if (rlon != rlon_old) {
01595           if ((++topo_nx) >= EX)
01596             ERRMSG("Too many longitudes!");
01597           topo_ny = 0;
```

```
01598         }
01599         rlon_old = rlon;
01600         topo_lon[topo_nx] = rlon;
01601         topo_lat[topo_ny] = rlat;
01602         topo_z[topo_nx][topo_ny] = rz;
01603         if ((++topo_ny) >= EY)
01604           ERRMSG("Too many latitudes!");
01605       }
01606     if ((++topo_nx) >= EX)
01607       ERRMSG("Too many longitudes!");
01608
01609     /* Close file... */
01610     fclose(in);
01611
01612     /* Check grid spacing... */
01613     if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
01614         || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
01615       printf("Warning: Grid spacing does not match!\n");
01616   }
01617
01618   /* Apply hydrostatic equation to calculate geopotential heights... */
01619 #pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
01620   for (ix = 0; ix < met->nx; ix++)
01621     for (iy = 0; iy < met->ny; iy++) {
01622
01623       /* Get surface height... */
01624       lon = met->lon[ix];
01625       if (lon < topo_lon[0])
01626         lon += 360;
01627       else if (lon > topo_lon[topo_nx - 1])
01628         lon -= 360;
01629       lat = met->lat[iy];
01630       tx = locate_reg(topo_lon, topo_nx, lon);
01631       ty = locate_reg(topo_lat, topo_ny, lat);
01632       z0 = LIN(topo_lon[tx], topo_z[tx][ty],
01633                topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
01634       z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
01635                topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
01636       z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);
01637
01638       /* Find surface pressure level... */
01639       ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
01640
01641       /* Get surface temperature... */
01642       ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
01643                met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
01644
01645       /* Upper part of profile... */
01646       met->z[ix][iy][ip0 + 1]
01647         = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
01648                    * log(met->ps[ix][iy] / met->p[ip0 + 1]));
01649       for (ip = ip0 + 2; ip < met->np; ip++)
01650         met->z[ix][iy][ip]
01651           = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
01652                      * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
01653                      * log(met->p[ip - 1] / met->p[ip]));
01654     }
01655
01656   /* Smooth fields... */
01657   for (ip = 0; ip < met->np; ip++) {
01658
01659     /* Median filter... */
01660 #pragma omp parallel for default(shared) private(ix,iy,n,ix2,ix3,iy2,data)
01661     for (ix = 0; ix < met->nx; ix++)
01662       for (iy = 0; iy < met->ny; iy++) {
01663         n = 0;
01664         for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
01665           ix3 = ix2;
01666           if (ix3 < 0)
01667             ix3 += met->nx;
01668           if (ix3 >= met->nx)
01669             ix3 -= met->nx;
01670           for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
01671                iy2++)
01672             if (gsl_finite(met->z[ix3][iy2][ip])) {
01673               data[n] = met->z[ix3][iy2][ip];
01674               n++;
01675             }
01676         }
01677         if (n > 0) {
01678           gsl_sort(data, 1, (size_t) n);
01679           help[ix][iy] = (float)
01680             gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
01681         } else
01682           help[ix][iy] = GSL_NAN;
01683       }
01684
```

```
01685     /* Copy data... */
01686 #pragma omp parallel for default(shared) private(ix,iy)
01687     for (ix = 0; ix < met->nx; ix++)
01688       for (iy = 0; iy < met->ny; iy++)
01689         met->z[ix][iy][ip] = help[ix][iy];
01690   }
01691 }
```

Here is the call graph for this function:



5.21.2.22 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 1695 of file libtrac.c.

```
01701             {
01702
01703   float *help;
01704
01705   int ip, ix, iy, varid;
01706
01707   /* Check if variable exists... */
01708   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
01709     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
01710       return;
01711
01712   /* Allocate... */
01713   ALLOC(help, float, met->nx * met->ny * met->np);
01714
01715   /* Read data... */
01716   NC(nc_get_var_float(ncid, varid, help));
01717
01718   /* Copy and check data... */
01719 #pragma omp parallel for default(shared) private(ix,iy,ip)
01720   for (ix = 0; ix < met->nx; ix++)
01721     for (iy = 0; iy < met->ny; iy++)
01722       for (ip = 0; ip < met->np; ip++) {
01723         dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
01724         if (fabsf(dest[ix][iy][ip]) < 1e14f)
01725           dest[ix][iy][ip] *= scl;
01726         else
01727           dest[ix][iy][ip] = GSL_NAN;
01728       }
01729
01730   /* Free... */
01731   free(help);
01732 }
```

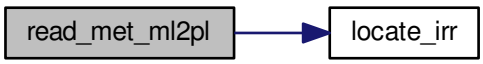
5.21.2.23 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 1736 of file libtrac.c.

```
01739                           {
01740
01741   double aux[EP], p[EP], pt;
01742
01743   int ip, ip2, ix, iy;
01744
01745   /* Loop over columns... */
01746 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
01747   for (ix = 0; ix < met->nx; ix++)
01748     for (iy = 0; iy < met->ny; iy++) {
01749
01750       /* Copy pressure profile... */
01751       for (ip = 0; ip < met->np; ip++)
01752         p[ip] = met->pl[ix][iy][ip];
01753
01754       /* Interpolate... */
01755       for (ip = 0; ip < ctl->met_np; ip++) {
01756         pt = ctl->met_p[ip];
01757         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
01758           pt = p[0];
01759         else if ((pt > p[met->np - 1] && p[1] > p[0])
01760                  || (pt < p[met->np - 1] && p[1] < p[0]))
01761           pt = p[met->np - 1];
01762         ip2 = locate_irr(p, met->np, pt);
01763         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
01764                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
01765       }
01766
01767       /* Copy data... */
01768       for (ip = 0; ip < ctl->met_np; ip++)
01769         var[ix][iy][ip] = (float) aux[ip];
01770     }
01771 }
```

Here is the call graph for this function:



5.21.2.24 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 1775 of file libtrac.c.

```
01776                 {
01777
01778   int ip, iy;
01779
01780   /* Check longitudes... */
01781   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
01782              + met->lon[1] - met->lon[0] - 360) < 0.01))
01783     return;
01784
01785   /* Increase longitude counter... */
```

```
01786   if ((++met->nx) > EX)
01787     ERRMSG("Cannot create periodic boundary conditions!");
01788
01789   /* Set longitude... */
01790   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01791
01792   /* Loop over latitudes and pressure levels... */
01793 #pragma omp parallel for default(shared) private(iy,ip)
01794   for (iy = 0; iy < met->ny; iy++) {
01795     met->ps[met->nx - 1][iy] = met->ps[0][iy];
01796     met->pt[met->nx - 1][iy] = met->pt[0][iy];
01797     for (ip = 0; ip < met->np; ip++) {
01798       met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
01799       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01800       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01801       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01802       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01803       met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
01804       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01805       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01806     }
01807   }
01808 }
```

5.21.2.25 void read_met_pv (met_t * *met*)

Calculate potential vorticity.

Definition at line 1812 of file libtrac.c.

```
01813               {
01814
01815   double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
01816     dtdp, dudp, dvdp, latr, vort, pows[EP];
01817
01818   int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
01819
01820   /* Set powers... */
01821   for (ip = 0; ip < met->np; ip++)
01822     pows[ip] = pow(1000. / met->p[ip], 0.286);
01823
01824   /* Loop over grid points... */
01825 #pragma omp parallel for default(shared)
       private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
01826   for (ix = 0; ix < met->nx; ix++) {
01827
01828     /* Set indices... */
01829     ix0 = GSL_MAX(ix - 1, 0);
01830     ix1 = GSL_MIN(ix + 1, met->nx - 1);
01831
01832     /* Loop over grid points... */
01833     for (iy = 0; iy < met->ny; iy++) {
01834
01835       /* Set indices... */
01836       iy0 = GSL_MAX(iy - 1, 0);
01837       iy1 = GSL_MIN(iy + 1, met->ny - 1);
01838
01839       /* Set auxiliary variables... */
01840       latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
01841       dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
01842       dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
01843       c0 = cos(met->lat[iy0] / 180. * M_PI);
01844       c1 = cos(met->lat[iy1] / 180. * M_PI);
01845       cr = cos(latr / 180. * M_PI);
01846       vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
01847
01848       /* Loop over grid points... */
01849       for (ip = 0; ip < met->np; ip++) {
01850
01851         /* Get gradients in longitude... */
01852         dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
01853         dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
01854
01855         /* Get gradients in latitude... */
01856         dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
01857         dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
01858
01859         /* Set indices... */
01860         ip0 = GSL_MAX(ip - 1, 0);
```

```
01861         ip1 = GSL_MIN(ip + 1, met->np - 1);
01862
01863         /* Get gradients in pressure... */
01864         dp0 = 100. * (met->p[ip] - met->p[ip0]);
01865         dp1 = 100. * (met->p[ip1] - met->p[ip]);
01866         if (ip != ip0 && ip != ip1) {
01867           denom = dp0 * dp1 * (dp0 + dp1);
01868           dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
01869                   - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
01870                   + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
01871             / denom;
01872           dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
01873                   - dp1 * dp1 * met->u[ix][iy][ip0]
01874                   + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
01875             / denom;
01876           dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
01877                   - dp1 * dp1 * met->v[ix][iy][ip0]
01878                   + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
01879             / denom;
01880         } else {
01881           denom = dp0 + dp1;
01882           dtdp =
01883             (met->t[ix][iy][ip1] * pows[ip1] -
01884              met->t[ix][iy][ip0] * pows[ip0]) / denom;
01885           dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
01886           dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
01887         }
01888
01889         /* Calculate PV... */
01890         met->pv[ix][iy][ip] = (float)
01891           (1e6 * G0 *
01892            (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
01893       }
01894     }
01895   }
01896 }
```

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

Downsampling of meteorological data.

Definition at line 1900 of file libtrac.c.

```
01902               {
01903
01904   met_t *help;
01905
01906   float w, wsum;
01907
01908   int ip, ip2, ix, ix2, ix3, iy, iy2;
01909
01910   /* Check parameters... */
01911   if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
01912       && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
01913     return;
01914
01915   /* Allocate... */
01916   ALLOC(help, met_t, 1);
01917
01918   /* Copy data... */
01919   help->nx = met->nx;
01920   help->ny = met->ny;
01921   help->np = met->np;
01922   memcpy(help->lon, met->lon, sizeof(met->lon));
01923   memcpy(help->lat, met->lat, sizeof(met->lat));
01924   memcpy(help->p, met->p, sizeof(met->p));
01925
01926   /* Smoothing... */
01927   for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
01928     for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
01929       for (ip = 0; ip < met->np; ip += ctl->met_dp) {
01930         help->ps[ix][iy] = 0;
01931         help->pt[ix][iy] = 0;
01932         help->z[ix][iy][ip] = 0;
01933         help->t[ix][iy][ip] = 0;
01934         help->u[ix][iy][ip] = 0;
01935         help->v[ix][iy][ip] = 0;
01936         help->w[ix][iy][ip] = 0;
01937         help->pv[ix][iy][ip] = 0;
01938         help->h2o[ix][iy][ip] = 0;
```

```
01939         help->o3[ix][iy][ip] = 0;
01940         wsum = 0;
01941         for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
01942           ix3 = ix2;
01943           if (ix3 < 0)
01944             ix3 += met->nx;
01945           else if (ix3 >= met->nx)
01946             ix3 -= met->nx;
01947
01948           for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
01949                iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
01950             for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
01951                  ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
01952               w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
01953                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
01954                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
01955               help->ps[ix][iy] += w * met->ps[ix3][iy2];
01956               help->pt[ix][iy] += w * met->pt[ix3][iy2];
01957               help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
01958               help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
01959               help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
01960               help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
01961               help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
01962               help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
01963               help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
01964               help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
01965               wsum += w;
01966             }
01967         }
01968         help->ps[ix][iy] /= wsum;
01969         help->pt[ix][iy] /= wsum;
01970         help->t[ix][iy][ip] /= wsum;
01971         help->z[ix][iy][ip] /= wsum;
01972         help->u[ix][iy][ip] /= wsum;
01973         help->v[ix][iy][ip] /= wsum;
01974         help->w[ix][iy][ip] /= wsum;
01975         help->pv[ix][iy][ip] /= wsum;
01976         help->h2o[ix][iy][ip] /= wsum;
01977         help->o3[ix][iy][ip] /= wsum;
01978       }
01979     }
01980   }
01981
01982   /* Downsampling... */
01983   met->nx = 0;
01984   for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
01985     met->lon[met->nx] = help->lon[ix];
01986     met->ny = 0;
01987     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
01988       met->lat[met->ny] = help->lat[iy];
01989       met->ps[met->nx][met->ny] = help->ps[ix][iy];
01990       met->pt[met->nx][met->ny] = help->pt[ix][iy];
01991       met->np = 0;
01992       for (ip = 0; ip < help->np; ip += ctl->met_dp) {
01993         met->p[met->np] = help->p[ip];
01994         met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
01995         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
01996         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
01997         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
01998         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
01999         met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
02000         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
02001         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
02002         met->np++;
02003       }
02004       met->ny++;
02005     }
02006     met->nx++;
02007   }
02008
02009   /* Free... */
02010   free(help);
02011 }
```

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

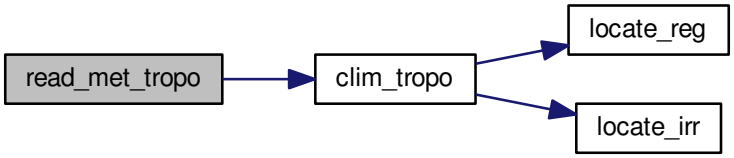
Calculate tropopause pressure.

Definition at line 2015 of file libtrac.c.

```
02017               {
02018
02019     gsl_interp_accel *acc;
02020
02021     gsl_spline *spline;
02022
02023     double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
02024       z[400], z2[400];
02025
02026     int found, ix, iy, iz, iz2;
02027
02028     /* Allocate... */
02029     acc = gsl_interp_accel_alloc();
02030     spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);
02031
02032     /* Get altitude and pressure profiles... */
02033     for (iz = 0; iz < met->np; iz++)
02034       z[iz] = Z(met->p[iz]);
02035     for (iz = 0; iz <= 170; iz++) {
02036       z2[iz] = 4.5 + 0.1 * iz;
02037       p2[iz] = P(z2[iz]);
02038     }
02039
02040     /* Do not calculate tropopause... */
02041     if (ctl->met_tropo == 0)
02042       for (ix = 0; ix < met->nx; ix++)
02043         for (iy = 0; iy < met->ny; iy++)
02044           met->pt[ix][iy] = GSL_NAN;
02045
02046     /* Use tropopause climatology... */
02047     else if (ctl->met_tropo == 1)
02048       for (ix = 0; ix < met->nx; ix++)
02049         for (iy = 0; iy < met->ny; iy++)
02050           met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);
02051
02052     /* Use cold point... */
02053     else if (ctl->met_tropo == 2) {
02054
02055       /* Loop over grid points... */
02056       for (ix = 0; ix < met->nx; ix++)
02057         for (iy = 0; iy < met->ny; iy++) {
02058
02059           /* Interpolate temperature profile... */
02060           for (iz = 0; iz < met->np; iz++)
02061             t[iz] = met->t[ix][iy][iz];
02062           gsl_spline_init(spline, z, t, (size_t) met->np);
02063           for (iz = 0; iz <= 170; iz++)
02064             t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02065
02066           /* Find minimum... */
02067           iz = (int) gsl_stats_min_index(t2, 1, 171);
02068           if (iz <= 0 || iz >= 170)
02069             met->pt[ix][iy] = GSL_NAN;
02070           else
02071             met->pt[ix][iy] = p2[iz];
02072         }
02073     }
02074
02075     /* Use WMO definition... */
02076     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
02077
02078       /* Loop over grid points... */
02079       for (ix = 0; ix < met->nx; ix++)
02080         for (iy = 0; iy < met->ny; iy++) {
02081
02082           /* Interpolate temperature profile... */
02083           for (iz = 0; iz < met->np; iz++)
02084             t[iz] = met->t[ix][iy][iz];
02085           gsl_spline_init(spline, z, t, (size_t) met->np);
02086           for (iz = 0; iz <= 160; iz++)
02087             t2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02088
02089           /* Find 1st tropopause... */
02090           met->pt[ix][iy] = GSL_NAN;
02091           for (iz = 0; iz <= 140; iz++) {
02092             found = 1;
02093             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02094               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02095                   / log(p2[iz2] / p2[iz]) > 2.0) {
02096                 found = 0;
02097                 break;
02098               }
02099             if (found) {
02100               if (iz > 0 && iz < 140)
02101                 met->pt[ix][iy] = p2[iz];
02102               break;
02103             }
```

```
02104         }
02105
02106         /* Find 2nd tropopause... */
02107         if (ctl->met_tropo == 4) {
02108           met->pt[ix][iy] = GSL_NAN;
02109           for (; iz <= 140; iz++) {
02110             found = 1;
02111             for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
02112               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02113                   / log(p2[iz2] / p2[iz]) < 3.0) {
02114                 found = 0;
02115                 break;
02116               }
02117             if (found)
02118               break;
02119           }
02120           for (; iz <= 140; iz++) {
02121             found = 1;
02122             for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
02123               if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
02124                   / log(p2[iz2] / p2[iz]) > 2.0) {
02125                 found = 0;
02126                 break;
02127               }
02128             if (found) {
02129               if (iz > 0 && iz < 140)
02130                 met->pt[ix][iy] = p2[iz];
02131               break;
02132             }
02133           }
02134         }
02135       }
02136   }
02137
02138   /* Use dynamical tropopause... */
02139   else if (ctl->met_tropo == 5) {
02140
02141     /* Loop over grid points... */
02142     for (ix = 0; ix < met->nx; ix++)
02143       for (iy = 0; iy < met->ny; iy++) {
02144
02145         /* Interpolate potential vorticity profile... */
02146         for (iz = 0; iz < met->np; iz++)
02147           pv[iz] = met->pv[ix][iy][iz];
02148         gsl_spline_init(spline, z, pv, (size_t) met->np);
02149         for (iz = 0; iz <= 160; iz++)
02150           pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02151
02152         /* Interpolate potential temperature profile... */
02153         for (iz = 0; iz < met->np; iz++)
02154           th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
02155         gsl_spline_init(spline, z, th, (size_t) met->np);
02156         for (iz = 0; iz <= 160; iz++)
02157           th2[iz] = gsl_spline_eval(spline, z2[iz], acc);
02158
02159         /* Find dynamical tropopause 3.5 PVU + 380 K */
02160         met->pt[ix][iy] = GSL_NAN;
02161         for (iz = 0; iz <= 160; iz++)
02162           if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
02163             if (iz > 0 && iz < 160)
02164               met->pt[ix][iy] = p2[iz];
02165             break;
02166           }
02167       }
02168   }
02169
02170   else
02171     ERRMSG("Cannot calculate tropopause!");
02172
02173   /* Free... */
02174   gsl_spline_free(spline);
02175   gsl_interp_accel_free(acc);
02176 }
```

Here is the call graph for this function:



5.21.2.28 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 2180 of file libtrac.c.

```
02187               {
02188
02189   FILE *in = NULL;
02190
02191   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
02192     msg[2 * LEN], rvarname[LEN], rval[LEN];
02193
02194   int contain = 0, i;
02195
02196   /* Open file... */
02197   if (filename[strlen(filename) - 1] != '-')
02198     if (!(in = fopen(filename, "r")))
02199       ERRMSG("Cannot open file!");
02200
02201   /* Set full variable name... */
02202   if (arridx >= 0) {
02203     sprintf(fullname1, "%s[%d]", varname, arridx);
02204     sprintf(fullname2, "%s[*]", varname);
02205   } else {
02206     sprintf(fullname1, "%s", varname);
02207     sprintf(fullname2, "%s", varname);
02208   }
02209
02210   /* Read data... */
02211   if (in != NULL)
02212     while (fgets(line, LEN, in))
02213       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
02214         if (strcasecmp(rvarname, fullname1) == 0 ||
02215             strcasecmp(rvarname, fullname2) == 0) {
02216           contain = 1;
02217           break;
02218         }
02219   for (i = 1; i < argc - 1; i++)
02220     if (strcasecmp(argv[i], fullname1) == 0 ||
02221         strcasecmp(argv[i], fullname2) == 0) {
02222       sprintf(rval, "%s", argv[i + 1]);
02223       contain = 1;
02224       break;
02225     }
02226
02227   /* Close file... */
02228   if (in != NULL)
02229     fclose(in);
02230
02231   /* Check for missing variables... */
02232   if (!contain) {
02233     if (strlen(defvalue) > 0)
02234       sprintf(rval, "%s", defvalue);
02235     else {
02236       sprintf(msg, "Missing variable %s!\n", fullname1);
02237       ERRMSG(msg);
```

```
02238     }
02239   }
02240
02241   /* Write info... */
02242   printf("%s = %s\n", fullname1, rval);
02243
02244   /* Return values... */
02245   if (value != NULL)
02246     sprintf(value, "%s", rval);
02247   return atof(rval);
02248 }
```

5.21.2.29 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 2252 of file libtrac.c.

```
02260                 {
02261
02262   struct tm t0, t1;
02263
02264   t0.tm_year = 100;
02265   t0.tm_mon = 0;
02266   t0.tm_mday = 1;
02267   t0.tm_hour = 0;
02268   t0.tm_min = 0;
02269   t0.tm_sec = 0;
02270
02271   t1.tm_year = year - 1900;
02272   t1.tm_mon = mon - 1;
02273   t1.tm_mday = day;
02274   t1.tm_hour = hour;
02275   t1.tm_min = min;
02276   t1.tm_sec = sec;
02277
02278   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
02279 }
```

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

Measure wall-clock time.

Definition at line 2283 of file libtrac.c.

```
02286             {
02287
02288   static double starttime[NTIMER], runtime[NTIMER];
02289
02290   /* Check id... */
02291   if (id < 0 || id >= NTIMER)
02292     ERRMSG("Too many timers!");
02293
02294   /* Start timer... */
02295   if (mode == 1) {
02296     if (starttime[id] <= 0)
02297       starttime[id] = omp_get_wtime();
02298     else
02299       ERRMSG("Timer already started!");
02300   }
02301
02302   /* Stop timer... */
02303   else if (mode == 2) {
02304     if (starttime[id] > 0) {
02305       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
02306       starttime[id] = -1;
02307     }
02308   }
02309
02310   /* Print timer... */
02311   else if (mode == 3) {
02312     printf("%s = %.3f s\n", name, runtime[id]);
02313     runtime[id] = 0;
02314   }
02315 }
```

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

Write atmospheric data.

Definition at line 2319 of file libtrac.c.

```
02323           {
02324
02325   FILE *in, *out;
02326
02327   char line[LEN];
02328
02329   double r, t0, t1;
02330
02331   int ip, iq, year, mon, day, hour, min, sec;
02332
02333   /* Set time interval for output... */
02334   t0 = t - 0.5 * ctl->dt_mod;
02335   t1 = t + 0.5 * ctl->dt_mod;
02336
02337   /* Write info... */
02338   printf("Write atmospheric data: %s\n", filename);
02339
02340   /* Write ASCII data... */
02341   if (ctl->atm_type == 0) {
02342
02343     /* Check if gnuplot output is requested... */
02344     if (ctl->atm_gpfile[0] != '-') {
02345
02346       /* Create gnuplot pipe... */
02347       if (!(out = popen("gnuplot", "w")))
02348         ERRMSG("Cannot create pipe to gnuplot!");
02349
02350       /* Set plot filename... */
02351       fprintf(out, "set out \"%s.png\"\n", filename);
02352
02353       /* Set time string... */
02354       jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02355       fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02356               year, mon, day, hour, min);
02357
02358       /* Dump gnuplot file to pipe... */
02359       if (!(in = fopen(ctl->atm_gpfile, "r")))
02360         ERRMSG("Cannot open file!");
02361       while (fgets(line, LEN, in))
02362         fprintf(out, "%s", line);
02363       fclose(in);
02364     }
02365
02366     else {
02367
02368       /* Create file... */
02369       if (!(out = fopen(filename, "w")))
02370         ERRMSG("Cannot create file!");
02371     }
02372
02373     /* Write header... */
02374     fprintf(out,
02375             "# $1 = time [s]\n"
02376             "# $2 = altitude [km]\n"
02377             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02378     for (iq = 0; iq < ctl->nq; iq++)
02379       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
02380               ctl->qnt_unit[iq]);
02381     fprintf(out, "\n");
02382
02383     /* Write data... */
02384     for (ip = 0; ip < atm->np; ip++) {
02385
02386       /* Check time... */
02387       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
02388         continue;
02389
02390       /* Write output... */
02391       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
02392               atm->lon[ip], atm->lat[ip]);
02393       for (iq = 0; iq < ctl->nq; iq++) {
02394         fprintf(out, " ");
02395         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02396       }
02397       fprintf(out, "\n");
02398     }
02399
```

```
02400     /* Close file... */
02401     fclose(out);
02402   }
02403
02404   /* Write binary data... */
02405   else if (ctl->atm_type == 1) {
02406
02407     /* Create file... */
02408     if (!(out = fopen(filename, "w")))
02409       ERRMSG("Cannot create file!");
02410
02411     /* Write data... */
02412     FWRITE(&atm->np, int,
02413            1,
02414            out);
02415     FWRITE(atm->time, double,
02416              (size_t) atm->np,
02417            out);
02418     FWRITE(atm->p, double,
02419              (size_t) atm->np,
02420            out);
02421     FWRITE(atm->lon, double,
02422              (size_t) atm->np,
02423            out);
02424     FWRITE(atm->lat, double,
02425              (size_t) atm->np,
02426            out);
02427     for (iq = 0; iq < ctl->nq; iq++)
02428       FWRITE(atm->q[iq], double,
02429                (size_t) atm->np,
02430              out);
02431
02432     /* Close file... */
02433     fclose(out);
02434   }
02435
02436   /* Error... */
02437   else
02438     ERRMSG("Atmospheric data type not supported!");
02439 }
```

Here is the call graph for this function:

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

Write CSI data.

Definition at line 2443 of file libtrac.c.

```
02447             {
02448
02449   static FILE *in, *out;
02450
02451   static char line[LEN];
02452
02453   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
02454     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
02455
02456   static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
02457
02458   /* Init... */
02459   if (t == ctl->t_start) {
```

```
02460
02461     /* Check quantity index for mass... */
02462     if (ctl->qnt_m < 0)
02463       ERRMSG("Need quantity mass!");
02464
02465     /* Open observation data file... */
02466     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
02467     if (!(in = fopen(ctl->csi_obsfile, "r")))
02468       ERRMSG("Cannot open file!");
02469
02470     /* Create new file... */
02471     printf("Write CSI data: %s\n", filename);
02472     if (!(out = fopen(filename, "w")))
02473       ERRMSG("Cannot create file!");
02474
02475     /* Write header... */
02476     fprintf(out,
02477             "# $1 = time [s]\n"
02478             "# $2 = number of hits (cx)\n"
02479             "# $3 = number of misses (cy)\n"
02480             "# $4 = number of false alarms (cz)\n"
02481             "# $5 = number of observations (cx + cy)\n"
02482             "# $6 = number of forecasts (cx + cz)\n"
02483             "# $7 = bias (forecasts/observations) [%%]\n"
02484             "# $8 = probability of detection (POD) [%%]\n"
02485             "# $9 = false alarm rate (FAR) [%%]\n"
02486             "# $10 = critical success index (CSI) [%%]\n\n");
02487   }
02488
02489   /* Set time interval... */
02490   t0 = t - 0.5 * ctl->dt_mod;
02491   t1 = t + 0.5 * ctl->dt_mod;
02492
02493   /* Initialize grid cells... */
02494 #pragma omp parallel for default(shared) private(ix,iy,iz)
02495   for (ix = 0; ix < ctl->csi_nx; ix++)
02496     for (iy = 0; iy < ctl->csi_ny; iy++)
02497       for (iz = 0; iz < ctl->csi_nz; iz++)
02498         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
02499
02500   /* Read observation data... */
02501   while (fgets(line, LEN, in)) {
02502
02503     /* Read data... */
02504     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02505         5)
02506       continue;
02507
02508     /* Check time... */
02509     if (rt < t0)
02510       continue;
02511     if (rt > t1)
02512       break;
02513
02514     /* Calculate indices... */
02515     ix = (int) ((rlon - ctl->csi_lon0)
02516                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02517     iy = (int) ((rlat - ctl->csi_lat0)
02518                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02519     iz = (int) ((rz - ctl->csi_z0)
02520                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
02521
02522     /* Check indices... */
02523     if (ix < 0 || ix >= ctl->csi_nx ||
02524         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02525       continue;
02526
02527     /* Get mean observation index... */
02528     obsmean[ix][iy][iz] += robs;
02529     obscount[ix][iy][iz]++;
02530   }
02531
02532   /* Analyze model data... */
02533 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02534   for (ip = 0; ip < atm->np; ip++) {
02535
02536     /* Check time... */
02537     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02538       continue;
02539
02540     /* Get indices... */
02541     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
02542                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
02543     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
02544                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
02545     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
02546                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
```

```
02547
02548     /* Check indices... */
02549     if (ix < 0 || ix >= ctl->csi_nx ||
02550         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
02551       continue;
02552
02553     /* Get total mass in grid cell... */
02554     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02555   }
02556
02557   /* Analyze all grid cells... */
02558 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
02559   for (ix = 0; ix < ctl->csi_nx; ix++)
02560     for (iy = 0; iy < ctl->csi_ny; iy++)
02561       for (iz = 0; iz < ctl->csi_nz; iz++) {
02562
02563         /* Calculate mean observation index... */
02564         if (obscount[ix][iy][iz] > 0)
02565           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
02566
02567         /* Calculate column density... */
02568         if (modmean[ix][iy][iz] > 0) {
02569           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
02570           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
02571           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
02572           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
02573             * cos(lat * M_PI / 180.);
02574           modmean[ix][iy][iz] /= (1e6 * area);
02575         }
02576
02577         /* Calculate CSI... */
02578         if (obscount[ix][iy][iz] > 0) {
02579           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02580               modmean[ix][iy][iz] >= ctl->csi_modmin)
02581             cx++;
02582           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
02583                    modmean[ix][iy][iz] < ctl->csi_modmin)
02584             cy++;
02585           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
02586                    modmean[ix][iy][iz] >= ctl->csi_modmin)
02587             cz++;
02588         }
02589       }
02590
02591   /* Write output... */
02592   if (fmod(t, ctl->csi_dt_out) == 0) {
02593
02594     /* Write... */
02595     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
02596             t, cx, cy, cz, cx + cy, cx + cz,
02597             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
02598             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
02599             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
02600             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
02601
02602     /* Set counters to zero... */
02603     cx = cy = cz = 0;
02604   }
02605
02606   /* Close file... */
02607   if (t == ctl->t_stop)
02608     fclose(out);
02609 }
```

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

Write ensemble data.

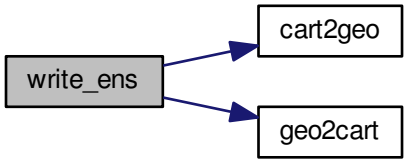
Definition at line 2613 of file libtrac.c.

```
02617             {
02618
02619   static FILE *out;
02620
02621   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
02622     t0, t1, x[NENS][3], xm[3];
02623
02624   static int ip, iq;
02625
02626   static size_t i, n;
```

```
02627
02628   /* Init... */
02629   if (t == ctl->t_start) {
02630
02631     /* Check quantities... */
02632     if (ctl->qnt_ens < 0)
02633       ERRMSG("Missing ensemble IDs!");
02634
02635     /* Create new file... */
02636     printf("Write ensemble data: %s\n", filename);
02637     if (!(out = fopen(filename, "w")))
02638       ERRMSG("Cannot create file!");
02639
02640     /* Write header... */
02641     fprintf(out,
02642             "# $1 = time [s]\n"
02643             "# $2 = altitude [km]\n"
02644             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02645     for (iq = 0; iq < ctl->nq; iq++)
02646       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
02647               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02648     for (iq = 0; iq < ctl->nq; iq++)
02649       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
02650               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02651     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
02652   }
02653
02654   /* Set time interval... */
02655   t0 = t - 0.5 * ctl->dt_mod;
02656   t1 = t + 0.5 * ctl->dt_mod;
02657
02658   /* Init... */
02659   ens = GSL_NAN;
02660   n = 0;
02661
02662   /* Loop over air parcels... */
02663   for (ip = 0; ip < atm->np; ip++) {
02664
02665     /* Check time... */
02666     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02667       continue;
02668
02669     /* Check ensemble id... */
02670     if (atm->q[ctl->qnt_ens][ip] != ens) {
02671
02672       /* Write results... */
02673       if (n > 0) {
02674
02675         /* Get mean position... */
02676         xm[0] = xm[1] = xm[2] = 0;
02677         for (i = 0; i < n; i++) {
02678           xm[0] += x[i][0] / (double) n;
02679           xm[1] += x[i][1] / (double) n;
02680           xm[2] += x[i][2] / (double) n;
02681         }
02682         cart2geo(xm, &dummy, &lon, &lat);
02683         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
02684                 lat);
02685
02686         /* Get quantity statistics... */
02687         for (iq = 0; iq < ctl->nq; iq++) {
02688           fprintf(out, " ");
02689           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02690         }
02691         for (iq = 0; iq < ctl->nq; iq++) {
02692           fprintf(out, " ");
02693           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02694         }
02695         fprintf(out, " %lu\n", n);
02696       }
02697
02698       /* Init new ensemble... */
02699       ens = atm->q[ctl->qnt_ens][ip];
02700       n = 0;
02701     }
02702
02703     /* Save data... */
02704     p[n] = atm->p[ip];
02705     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
02706     for (iq = 0; iq < ctl->nq; iq++)
02707       q[iq][n] = atm->q[iq][ip];
02708     if ((++n) >= NENS)
02709       ERRMSG("Too many data points!");
02710   }
02711
02712   /* Write results... */
02713   if (n > 0) {
```

```
02714
02715     /* Get mean position... */
02716     xm[0] = xm[1] = xm[2] = 0;
02717     for (i = 0; i < n; i++) {
02718       xm[0] += x[i][0] / (double) n;
02719       xm[1] += x[i][1] / (double) n;
02720       xm[2] += x[i][2] / (double) n;
02721     }
02722     cart2geo(xm, &dummy, &lon, &lat);
02723     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
02724
02725     /* Get quantity statistics... */
02726     for (iq = 0; iq < ctl->nq; iq++) {
02727       fprintf(out, " ");
02728       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
02729     }
02730     for (iq = 0; iq < ctl->nq; iq++) {
02731       fprintf(out, " ");
02732       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
02733     }
02734     fprintf(out, " %lu\n", n);
02735   }
02736
02737   /* Close file... */
02738   if (t == ctl->t_stop)
02739     fclose(out);
02740 }
```

Here is the call graph for this function:



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

Write gridded data.

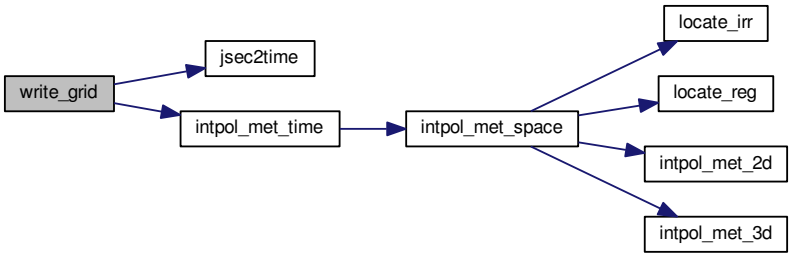
Definition at line 2744 of file libtrac.c.

```
02750             {
02751
02752   FILE *in, *out;
02753
02754   char line[LEN];
02755
02756   static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
02757     area, rho_air, press, temp, cd, vmr, t0, t1, r;
02758
02759   static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;
02760
02761   /* Check dimensions... */
02762   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
02763     ERRMSG("Grid dimensions too large!");
02764
02765   /* Set time interval for output... */
02766   t0 = t - 0.5 * ctl->dt_mod;
02767   t1 = t + 0.5 * ctl->dt_mod;
02768
02769   /* Set grid box size... */
02770   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
02771   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
```

```
02772   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
02773
02774   /* Initialize grid... */
02775 #pragma omp parallel for default(shared) private(ix,iy,iz)
02776   for (ix = 0; ix < ctl->grid_nx; ix++)
02777     for (iy = 0; iy < ctl->grid_ny; iy++)
02778       for (iz = 0; iz < ctl->grid_nz; iz++) {
02779         mass[ix][iy][iz] = 0;
02780         np[ix][iy][iz] = 0;
02781       }
02782
02783   /* Average data... */
02784 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
02785   for (ip = 0; ip < atm->np; ip++)
02786     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
02787
02788       /* Get index... */
02789       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
02790       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
02791       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
02792
02793       /* Check indices... */
02794       if (ix < 0 || ix >= ctl->grid_nx ||
02795           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
02796         continue;
02797
02798       /* Add mass... */
02799       if (ctl->qnt_m >= 0)
02800         mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
02801       np[ix][iy][iz]++;
02802     }
02803
02804   /* Check if gnuplot output is requested... */
02805   if (ctl->grid_gpfile[0] != '-') {
02806
02807     /* Write info... */
02808     printf("Plot grid data: %s.png\n", filename);
02809
02810     /* Create gnuplot pipe... */
02811     if (!(out = popen("gnuplot", "w")))
02812       ERRMSG("Cannot create pipe to gnuplot!");
02813
02814     /* Set plot filename... */
02815     fprintf(out, "set out \"%s.png\"\n", filename);
02816
02817     /* Set time string... */
02818     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02819     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
02820             year, mon, day, hour, min);
02821
02822     /* Dump gnuplot file to pipe... */
02823     if (!(in = fopen(ctl->grid_gpfile, "r")))
02824       ERRMSG("Cannot open file!");
02825     while (fgets(line, LEN, in))
02826       fprintf(out, "%s", line);
02827     fclose(in);
02828   }
02829
02830   else {
02831
02832     /* Write info... */
02833     printf("Write grid data: %s\n", filename);
02834
02835     /* Create file... */
02836     if (!(out = fopen(filename, "w")))
02837       ERRMSG("Cannot create file!");
02838   }
02839
02840   /* Write header... */
02841   fprintf(out,
02842           "# $1 = time [s]\n"
02843           "# $2 = altitude [km]\n"
02844           "# $3 = longitude [deg]\n"
02845           "# $4 = latitude [deg]\n"
02846           "# $5 = surface area [km^2]\n"
02847           "# $6 = layer width [km]\n"
02848           "# $7 = number of particles [1]\n"
02849           "# $8 = column density [kg/m^2]\n"
02850           "# $9 = volume mixing ratio [1]\n\n");
02851
02852   /* Write data... */
02853   for (ix = 0; ix < ctl->grid_nx; ix++) {
02854     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
02855       fprintf(out, "\n");
02856     for (iy = 0; iy < ctl->grid_ny; iy++) {
02857       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
02858         fprintf(out, "\n");
```

```
02859       for (iz = 0; iz < ctl->grid_nz; iz++)
02860         if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
02861
02862           /* Set coordinates... */
02863           z = ctl->grid_z0 + dz * (iz + 0.5);
02864           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
02865           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
02866
02867           /* Get pressure and temperature... */
02868           press = P(z);
02869           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
02870                           NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);
02871
02872           /* Calculate surface area... */
02873           area = dlat * dlon * SQR(RE * M_PI / 180.)
02874             * cos(lat * M_PI / 180.);
02875
02876           /* Calculate column density... */
02877           cd = mass[ix][iy][iz] / (1e6 * area);
02878
02879           /* Calculate volume mixing ratio... */
02880           rho_air = 100. * press / (RA * temp);
02881           vmr = MA / ctl->molmass * mass[ix][iy][iz]
02882             / (rho_air * 1e6 * area * 1e3 * dz);
02883
02884           /* Write output... */
02885           fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
02886                   t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
02887         }
02888     }
02889   }
02890
02891   /* Close file... */
02892   fclose(out);
02893 }
```

Here is the call graph for this function:



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

Write profile data.

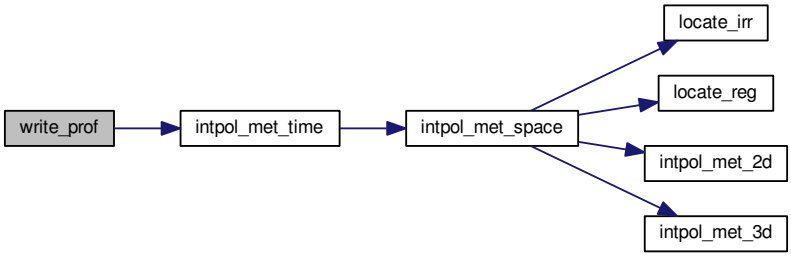
Definition at line 2897 of file libtrac.c.

```
02903             {
02904
02905   static FILE *in, *out;
02906
02907   static char line[LEN];
02908
02909   static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
02910     rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
02911     rho_air, vmr, h2o, o3;
02912
02913   static int obscount[GX][GY], ip, ix, iy, iz, okay;
02914
```

```
02915   /* Init... */
02916   if (t == ctl->t_start) {
02917
02918     /* Check quantity index for mass... */
02919     if (ctl->qnt_m < 0)
02920       ERRMSG("Need quantity mass!");
02921
02922     /* Check dimensions... */
02923     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
02924       ERRMSG("Grid dimensions too large!");
02925
02926     /* Open observation data file... */
02927     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
02928     if (!(in = fopen(ctl->prof_obsfile, "r")))
02929       ERRMSG("Cannot open file!");
02930
02931     /* Create new output file... */
02932     printf("Write profile data: %s\n", filename);
02933     if (!(out = fopen(filename, "w")))
02934       ERRMSG("Cannot create file!");
02935
02936     /* Write header... */
02937     fprintf(out,
02938             "# $1 = time [s]\n"
02939             "# $2 = altitude [km]\n"
02940             "# $3 = longitude [deg]\n"
02941             "# $4 = latitude [deg]\n"
02942             "# $5 = pressure [hPa]\n"
02943             "# $6 = temperature [K]\n"
02944             "# $7 = volume mixing ratio [1]\n"
02945             "# $8 = H2O volume mixing ratio [1]\n"
02946             "# $9 = O3 volume mixing ratio [1]\n"
02947             "# $10 = observed BT index (mean) [K]\n"
02948             "# $11 = observed BT index (sigma) [K]\n");
02949
02950     /* Set grid box size... */
02951     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
02952     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
02953     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
02954   }
02955
02956   /* Set time interval... */
02957   t0 = t - 0.5 * ctl->dt_mod;
02958   t1 = t + 0.5 * ctl->dt_mod;
02959
02960   /* Initialize... */
02961 #pragma omp parallel for default(shared) private(ix,iy,iz)
02962   for (ix = 0; ix < ctl->prof_nx; ix++)
02963     for (iy = 0; iy < ctl->prof_ny; iy++) {
02964       obsmean[ix][iy] = 0;
02965       obsmean2[ix][iy] = 0;
02966       obscount[ix][iy] = 0;
02967       for (iz = 0; iz < ctl->prof_nz; iz++)
02968         mass[ix][iy][iz] = 0;
02969     }
02970
02971   /* Read observation data... */
02972   while (fgets(line, LEN, in)) {
02973
02974     /* Read data... */
02975     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
02976         5)
02977       continue;
02978
02979     /* Check time... */
02980     if (rt < t0)
02981       continue;
02982     if (rt > t1)
02983       break;
02984
02985     /* Calculate indices... */
02986     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
02987     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
02988
02989     /* Check indices... */
02990     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
02991       continue;
02992
02993     /* Get mean observation index... */
02994     obsmean[ix][iy] += robs;
02995     obsmean2[ix][iy] += SQR(robs);
02996     obscount[ix][iy]++;
02997   }
02998
02999   /* Analyze model data... */
03000 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03001   for (ip = 0; ip < atm->np; ip++) {
```

```
03002
03003     /* Check time... */
03004     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03005       continue;
03006
03007     /* Get indices... */
03008     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
03009     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
03010     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
03011
03012     /* Check indices... */
03013     if (ix < 0 || ix >= ctl->prof_nx ||
03014         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
03015       continue;
03016
03017     /* Get total mass in grid cell... */
03018     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03019   }
03020
03021   /* Extract profiles... */
03022   for (ix = 0; ix < ctl->prof_nx; ix++)
03023     for (iy = 0; iy < ctl->prof_ny; iy++)
03024       if (obscount[ix][iy] > 0) {
03025
03026         /* Check profile... */
03027         okay = 0;
03028         for (iz = 0; iz < ctl->prof_nz; iz++)
03029           if (mass[ix][iy][iz] > 0) {
03030             okay = 1;
03031             break;
03032           }
03033         if (!okay)
03034           continue;
03035
03036         /* Write output... */
03037         fprintf(out, "\n");
03038
03039         /* Loop over altitudes... */
03040         for (iz = 0; iz < ctl->prof_nz; iz++) {
03041
03042           /* Set coordinates... */
03043           z = ctl->prof_z0 + dz * (iz + 0.5);
03044           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
03045           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
03046
03047           /* Get pressure and temperature... */
03048           press = P(z);
03049           intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
03050                           NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);
03051
03052           /* Calculate surface area... */
03053           area = dlat * dlon * SQR(M_PI * RE / 180.)
03054             * cos(lat * M_PI / 180.);
03055
03056           /* Calculate volume mixing ratio... */
03057           rho_air = 100. * press / (RA * temp);
03058           vmr = MA / ctl->molmass * mass[ix][iy][iz]
03059             / (rho_air * area * dz * 1e9);
03060
03061           /* Write output... */
03062           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n",
03063                   t, z, lon, lat, press, temp, vmr, h2o, o3,
03064                   obsmean[ix][iy] / obscount[ix][iy],
03065                   sqrt(obsmean2[ix][iy] / obscount[ix][iy]
03066                        - SQR(obsmean[ix][iy] / obscount[ix][iy])));
03067         }
03068       }
03069
03070   /* Close file... */
03071   if (t == ctl->t_stop)
03072     fclose(out);
03073 }
```

Here is the call graph for this function:



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

Write station data.

Definition at line 3077 of file libtrac.c.

```
03081            {
03082
03083   static FILE *out;
03084
03085   static double rmax2, t0, t1, x0[3], x1[3];
03086
03087   static int ip, iq;
03088
03089   /* Init... */
03090   if (t == ctl->t_start) {
03091
03092     /* Write info... */
03093     printf("Write station data: %s\n", filename);
03094
03095     /* Create new file... */
03096     if (!(out = fopen(filename, "w")))
03097       ERRMSG("Cannot create file!");
03098
03099     /* Write header... */
03100     fprintf(out,
03101             "# $1 = time [s]\n"
03102             "# $2 = altitude [km]\n"
03103             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03104     for (iq = 0; iq < ctl->nq; iq++)
03105       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
03106               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03107     fprintf(out, "\n");
03108
03109     /* Set geolocation and search radius... */
03110     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
03111     rmax2 = SQR(ctl->stat_r);
03112   }
03113
03114   /* Set time interval for output... */
03115   t0 = t - 0.5 * ctl->dt_mod;
03116   t1 = t + 0.5 * ctl->dt_mod;
03117
03118   /* Loop over air parcels... */
03119   for (ip = 0; ip < atm->np; ip++) {
03120
03121     /* Check time... */
03122     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03123       continue;
03124
03125     /* Check station flag... */
03126     if (ctl->qnt_stat >= 0)
03127       if (atm->q[ctl->qnt_stat][ip])
03128         continue;
03129
03130     /* Get Cartesian coordinates... */
```

```
03131     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
03132
03133     /* Check horizontal distance... */
03134     if (DIST2(x0, x1) > rmax2)
03135       continue;
03136
03137     /* Set station flag... */
03138     if (ctl->qnt_stat >= 0)
03139       atm->q[ctl->qnt_stat][ip] = 1;
03140
03141     /* Write data... */
03142     fprintf(out, "%.2f %g %g %g",
03143             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
03144     for (iq = 0; iq < ctl->nq; iq++) {
03145       fprintf(out, " ");
03146       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03147     }
03148     fprintf(out, "\n");
03149   }
03150
03151   /* Close file... */
03152   if (t == ctl->t_stop)
03153     fclose(out);
03154 }
```

Here is the call graph for this function:

5.22 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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00035 #include <ctype.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 MA 28.9644
00066
00068 #define P0 1013.25
00069
00071 #define RA 287.058
00072
00074 #define RI 8.3144598
00075
00077 #define RE 6367.421
00078
00079 /* ------------------------------------------------------------
00080    Dimensions...
00081    ------------------------------------------------------------ */
00082
00084 #define LEN 5000
00085
00087 #define NP 10000000
00088
00090 #define NQ 12
00091
00093 #define EP 112
00094
00096 #define EX 1201
00097
00099 #define EY 601
00100
00102 #define GX 720
00103
00105 #define GY 360
00106
00108 #define GZ 100
00109
00111 #define NENS 2000
00112
00114 #define NTHREADS 512
00115
00116 /* ------------------------------------------------------------
00117    Macros...
00118    ------------------------------------------------------------ */
00119
00121 #define ALLOC(ptr, type, n)                              \
00122   if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)      \
00123     ERRMSG("Out of memory!");
00124
00126 #define DEG2DX(dlon, lat)                                \
00127   ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00128
00130 #define DEG2DY(dlat)                                     \
00131   ((dlat) * M_PI * RE / 180.)
00132
00134 #define DP2DZ(dp, p)                                     \
00135   (- (dp) * H0 / (p))
00136
00138 #define DX2DEG(dx, lat)                                  \
00139   (((lat) < -89.999 || (lat) > 89.999) ? 0               \
00140    : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))
00141
00143 #define DY2DEG(dy)                                       \
00144   ((dy) * 180. / (M_PI * RE))
00145
00147 #define DZ2DP(dz, p)                                     \
00148   (-(dz) * (p) / H0)
00149
00151 #define DIST(a, b) sqrt(DIST2(a, b))
00152
00154 #define DIST2(a, b)                                                  \
00155   ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00156
00158 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00159
00161 #define ERRMSG(msg) {                                            \
00162     printf("\nError (%s, %s, l%d): %s\n\n",                      \
00163            __FILE__, __func__, __LINE__, msg);                   \
00164     exit(EXIT_FAILURE);                                          \
00165   }
00166
00168 #define FREAD(ptr, type, size, out) {                            \
00169     if(fread(ptr, sizeof(type), size, out)!=size)                \
```

```
00170       ERRMSG("Error while reading!");                                  \
00171   }
00172
00174 #define FWRITE(ptr, type, size, out) {                                 \
00175     if(fwrite(ptr, sizeof(type), size, out)!=size)                     \
00176       ERRMSG("Error while writing!");                                  \
00177   }
00178
00180 #define LIN(x0, y0, x1, y1, x)                  \
00181   ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00182
00184 #define NC(cmd) {                                   \
00185     if((cmd)!=NC_NOERR)                             \
00186       ERRMSG(nc_strerror(cmd));                     \
00187   }
00188
00190 #define NORM(a) sqrt(DOTP(a, a))
00191
00193 #define PRINT(format, var)                                             \
00194   printf("Print (%s, %s, l%d): %s= "format"\n",                        \
00195          __FILE__, __func__, __LINE__, #var, var);
00196
00198 #define P(z) (P0*exp(-(z)/H0))
00199
00201 #define SQR(x) ((x)*(x))
00202
00204 #define THETA(p, t) ((t)*pow(1000./(p), 0.286))
00205
00207 #define TOK(line, tok, format, var) {                                  \
00208     if(((tok)=strtok((line), " \t"))) {                                \
00209       if(sscanf(tok, format, &(var))!=1) continue;                     \
00210     } else ERRMSG("Error while reading!");                             \
00211   }
00212
00214 #define Z(p) (H0*log(P0/(p)))
00215
00216 /* ------------------------------------------------------------
00217    Timers...
00218    ------------------------------------------------------------ */
00219
00221 #define START_TIMER(id) timer(#id, id, 1)
00222
00224 #define STOP_TIMER(id) timer(#id, id, 2)
00225
00227 #define PRINT_TIMER(id) timer(#id, id, 3)
00228
00230 #define NTIMER 12
00231
00233 #define TIMER_TOTAL 0
00234
00236 #define TIMER_INIT 1
00237
00239 #define TIMER_INPUT 2
00240
00242 #define TIMER_OUTPUT 3
00243
00245 #define TIMER_ADVECT 4
00246
00248 #define TIMER_DECAY 5
00249
00251 #define TIMER_DIFFMESO 6
00252
00254 #define TIMER_DIFFTURB 7
00255
00257 #define TIMER_ISOSURF 8
00258
00260 #define TIMER_METEO 9
00261
00263 #define TIMER_POSITION 10
00264
00266 #define TIMER_SEDI 11
00267
00268 /* ------------------------------------------------------------
00269    Structs...
00270    ------------------------------------------------------------ */
00271
00273 typedef struct {
00274
00276   int nq;
00277
00279   char qnt_name[NQ][LEN];
00280
00282   char qnt_unit[NQ][LEN];
00283
00285   char qnt_format[NQ][LEN];
00286
00288   int qnt_ens;
```

```
00289
00291   int qnt_m;
00292
00294   int qnt_rho;
00295
00297   int qnt_r;
00298
00300   int qnt_ps;
00301
00303   int qnt_pt;
00304
00306   int qnt_z;
00307
00309   int qnt_p;
00310
00312   int qnt_t;
00313
00315   int qnt_u;
00316
00318   int qnt_v;
00319
00321   int qnt_w;
00322
00324   int qnt_h2o;
00325
00327   int qnt_o3;
00328
00330   int qnt_theta;
00331
00333   int qnt_vh;
00334
00336   int qnt_vz;
00337
00339   int qnt_pv;
00340
00342   int qnt_tice;
00343
00345   int qnt_tsts;
00346
00348   int qnt_tnat;
00349
00351   int qnt_stat;
00352
00354   int direction;
00355
00357   double t_start;
00358
00360   double t_stop;
00361
00363   double dt_mod;
00364
00366   double dt_met;
00367
00369   int met_dx;
00370
00372   int met_dy;
00373
00375   int met_dp;
00376
00378   int met_sx;
00379
00381   int met_sy;
00382
00384   int met_sp;
00385
00387   int met_np;
00388
00390   double met_p[EP];
00391
00394   int met_tropo;
00395
00397   char met_geopot[LEN];
00398
00400   double met_dt_out;
00401
00403   char met_stage[LEN];
00404
00407   int isosurf;
00408
00410   char balloon[LEN];
00411
00413   double turb_dx_trop;
00414
00416   double turb_dx_strat;
00417
00419   double turb_dz_trop;
00420
```

```
00422   double turb_dz_strat;
00423
00425   double turb_mesox;
00426
00428   double turb_mesoz;
00429
00431   double molmass;
00432
00434   double tdec_trop;
00435
00437   double tdec_strat;
00438
00440   double psc_h2o;
00441
00443   double psc_hno3;
00444
00446   char atm_basename[LEN];
00447
00449   char atm_gpfile[LEN];
00450
00452   double atm_dt_out;
00453
00455   int atm_filter;
00456
00458   int atm_type;
00459
00461   char csi_basename[LEN];
00462
00464   double csi_dt_out;
00465
00467   char csi_obsfile[LEN];
00468
00470   double csi_obsmin;
00471
00473   double csi_modmin;
00474
00476   int csi_nz;
00477
00479   double csi_z0;
00480
00482   double csi_z1;
00483
00485   int csi_nx;
00486
00488   double csi_lon0;
00489
00491   double csi_lon1;
00492
00494   int csi_ny;
00495
00497   double csi_lat0;
00498
00500   double csi_lat1;
00501
00503   char grid_basename[LEN];
00504
00506   char grid_gpfile[LEN];
00507
00509   double grid_dt_out;
00510
00512   int grid_sparse;
00513
00515   int grid_nz;
00516
00518   double grid_z0;
00519
00521   double grid_z1;
00522
00524   int grid_nx;
00525
00527   double grid_lon0;
00528
00530   double grid_lon1;
00531
00533   int grid_ny;
00534
00536   double grid_lat0;
00537
00539   double grid_lat1;
00540
00542   char prof_basename[LEN];
00543
00545   char prof_obsfile[LEN];
00546
00548   int prof_nz;
00549
00551   double prof_z0;
```

```
00552
00554   double prof_z1;
00555
00557   int prof_nx;
00558
00560   double prof_lon0;
00561
00563   double prof_lon1;
00564
00566   int prof_ny;
00567
00569   double prof_lat0;
00570
00572   double prof_lat1;
00573
00575   char ens_basename[LEN];
00576
00578   char stat_basename[LEN];
00579
00581   double stat_lon;
00582
00584   double stat_lat;
00585
00587   double stat_r;
00588
00589 } ctl_t;
00590
00592 typedef struct {
00593
00595   int np;
00596
00598   double time[NP];
00599
00601   double p[NP];
00602
00604   double lon[NP];
00605
00607   double lat[NP];
00608
00610   double q[NQ][NP];
00611
00613   float up[NP];
00614
00616   float vp[NP];
00617
00619   float wp[NP];
00620
00622   double cache_time[EX][EY][EP];
00623
00625   float cache_usig[EX][EY][EP];
00626
00628   float cache_vsig[EX][EY][EP];
00629
00631   float cache_wsig[EX][EY][EP];
00632
00633 } atm_t;
00634
00636 typedef struct {
00637
00639   double time;
00640
00642   int nx;
00643
00645   int ny;
00646
00648   int np;
00649
00651   double lon[EX];
00652
00654   double lat[EY];
00655
00657   double p[EP];
00658
00660   double ps[EX][EY];
00661
00663   double pt[EX][EY];
00664
00666   float z[EX][EY][EP];
00667
00669   float t[EX][EY][EP];
00670
00672   float u[EX][EY][EP];
00673
00675   float v[EX][EY][EP];
00676
00678   float w[EX][EY][EP];
00679
```

```
00681   float pv[EX][EY][EP];
00682
00684   float h2o[EX][EY][EP];
00685
00687   float o3[EX][EY][EP];
00688
00690   float pl[EX][EY][EP];
00691
00692 } met_t;
00693
00694 /* ------------------------------------------------------------
00695    Functions...
00696    ------------------------------------------------------------ */
00697
00699 void cart2geo(
00700   double *x,
00701   double *z,
00702   double *lon,
00703   double *lat);
00704
00706 double clim_hno3(
00707   double t,
00708   double lat,
00709   double p);
00710
00712 double clim_tropo(
00713   double t,
00714   double lat);
00715
00717 void day2doy(
00718   int year,
00719   int mon,
00720   int day,
00721   int *doy);
00722
00724 void doy2day(
00725   int year,
00726   int doy,
00727   int *mon,
00728   int *day);
00729
00731 void geo2cart(
00732   double z,
00733   double lon,
00734   double lat,
00735   double *x);
00736
00738 void get_met(
00739   ctl_t * ctl,
00740   char *metbase,
00741   double t,
00742   met_t ** met0,
00743   met_t ** met1);
00744
00746 void get_met_help(
00747   double t,
00748   int direct,
00749   char *metbase,
00750   double dt_met,
00751   char *filename);
00752
00754 void get_met_replace(
00755   char *orig,
00756   char *search,
00757   char *repl);
00758
00760 void intpol_met_2d(
00761   double array[EX][EY],
00762   int ix,
00763   int iy,
00764   double wx,
00765   double wy,
00766   double *var);
00767
00769 void intpol_met_3d(
00770   float array[EX][EY][EP],
00771   int ip,
00772   int ix,
00773   int iy,
00774   double wp,
00775   double wx,
00776   double wy,
00777   double *var);
00778
00780 void intpol_met_space(
00781   met_t * met,
00782   double p,
```

```
00783   double lon,
00784   double lat,
00785   double *ps,
00786   double *pt,
00787   double *z,
00788   double *t,
00789   double *u,
00790   double *v,
00791   double *w,
00792   double *pv,
00793   double *h2o,
00794   double *o3);
00795
00797 void intpol_met_time(
00798   met_t * met0,
00799   met_t * met1,
00800   double ts,
00801   double p,
00802   double lon,
00803   double lat,
00804   double *ps,
00805   double *pt,
00806   double *z,
00807   double *t,
00808   double *u,
00809   double *v,
00810   double *w,
00811   double *pv,
00812   double *h2o,
00813   double *o3);
00814
00816 void jsec2time(
00817   double jsec,
00818   int *year,
00819   int *mon,
00820   int *day,
00821   int *hour,
00822   int *min,
00823   int *sec,
00824   double *remain);
00825
00827 int locate_irr(
00828   double *xx,
00829   int n,
00830   double x);
00831
00833 int locate_reg(
00834   double *xx,
00835   int n,
00836   double x);
00837
00839 int read_atm(
00840   const char *filename,
00841   ctl_t * ctl,
00842   atm_t * atm);
00843
00845 void read_ctl(
00846   const char *filename,
00847   int argc,
00848   char *argv[],
00849   ctl_t * ctl);
00850
00852 int read_met(
00853   ctl_t * ctl,
00854   char *filename,
00855   met_t * met);
00856
00858 void read_met_extrapolate(
00859   met_t * met);
00860
00862 void read_met_geopot(
00863   ctl_t * ctl,
00864   met_t * met);
00865
00867 void read_met_help(
00868   int ncid,
00869   char *varname,
00870   char *varname2,
00871   met_t * met,
00872   float dest[EX][EY][EP],
00873   float scl);
00874
00876 void read_met_ml2pl(
00877   ctl_t * ctl,
00878   met_t * met,
00879   float var[EX][EY][EP]);
00880
```

```
00882 void read_met_periodic(
00883   met_t * met);
00884
00886 void read_met_pv(
00887   met_t * met);
00888
00890 void read_met_sample(
00891   ctl_t * ctl,
00892   met_t * met);
00893
00895 void read_met_tropo(
00896   ctl_t * ctl,
00897   met_t * met);
00898
00900 double scan_ctl(
00901   const char *filename,
00902   int argc,
00903   char *argv[],
00904   const char *varname,
00905   int arridx,
00906   const char *defvalue,
00907   char *value);
00908
00910 void time2jsec(
00911   int year,
00912   int mon,
00913   int day,
00914   int hour,
00915   int min,
00916   int sec,
00917   double remain,
00918   double *jsec);
00919
00921 void timer(
00922   const char *name,
00923   int id,
00924   int mode);
00925
00927 void write_atm(
00928   const char *filename,
00929   ctl_t * ctl,
00930   atm_t * atm,
00931   double t);
00932
00934 void write_csi(
00935   const char *filename,
00936   ctl_t * ctl,
00937   atm_t * atm,
00938   double t);
00939
00941 void write_ens(
00942   const char *filename,
00943   ctl_t * ctl,
00944   atm_t * atm,
00945   double t);
00946
00948 void write_grid(
00949   const char *filename,
00950   ctl_t * ctl,
00951   met_t * met0,
00952   met_t * met1,
00953   atm_t * atm,
00954   double t);
00955
00957 void write_prof(
00958   const char *filename,
00959   ctl_t * ctl,
00960   met_t * met0,
00961   met_t * met1,
00962   atm_t * atm,
00963   double t);
00964
00966 void write_station(
00967   const char *filename,
00968   ctl_t * ctl,
00969   atm_t * atm,
00970   double t);
```

5.23 met_map.c File Reference

Extract global map from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.23.1 Detailed Description

Extract global map from meteorological data.

Definition in file met_map.c.

5.23.2 Function Documentation

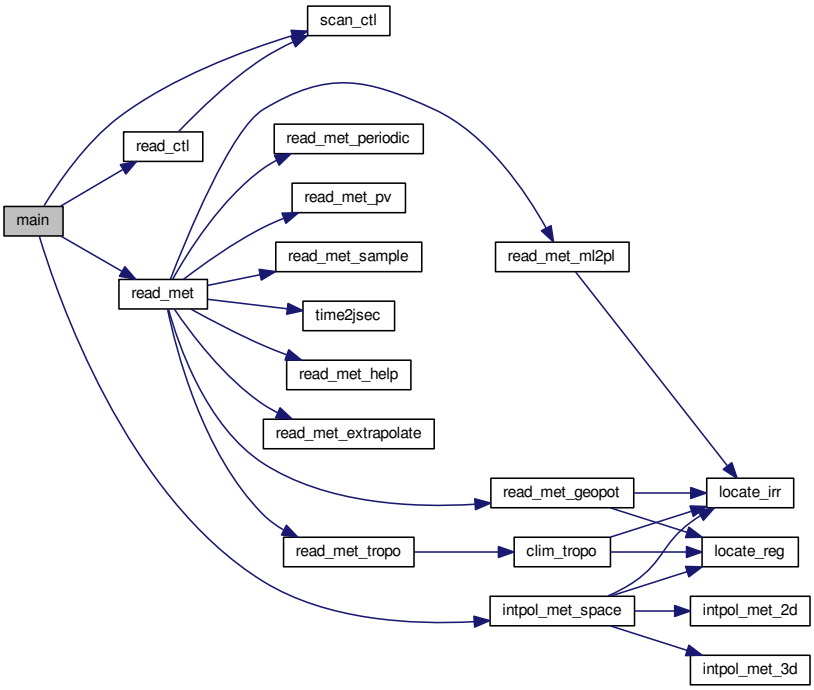
5.23.2.1 int main (int *argc*, char ∗ *argv[ ]*)

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 timem[EX][EY], p0, ps, psm[EX][EY], pt, ptm[EX][EY], t,
00038     tm[EX][EY], u, um[EX][EY], v, vm[EX][EY], w, wm[EX][EY], h2o,
00039     h2om[EX][EY], o3, o3m[EX][EY], z, zm[EX][EY], pv, pvm[EX][EY], zt,
00040     ztm[EX][EY], tt, ttm[EX][EY], lon, lon0, lon1, lons[EX], dlon, lat, lat0,
00041     lat1, lats[EY], dlat;
00042
00043   static int i, ix, iy, np[EX][EY], nx, ny;
00044
00045   /* Allocate... */
00046   ALLOC(met, met_t, 1);
00047
00048   /* Check arguments... */
00049   if (argc < 4)
00050     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00051
00052   /* Read control parameters... */
00053   read_ctl(argv[1], argc, argv, &ctl);
00054   p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "", NULL));
00055   lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00056   lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00057   dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00058   lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00059   lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00060   dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00061
00062   /* Loop over files... */
00063   for (i = 3; i < argc; i++) {
00064
00065     /* Read meteorological data... */
00066     if (!read_met(&ctl, argv[i], met))
00067       continue;
00068
00069     /* Set horizontal grid... */
00070     if (dlon <= 0)
00071       dlon = fabs(met->lon[1] - met->lon[0]);
00072     if (dlat <= 0)
00073       dlat = fabs(met->lat[1] - met->lat[0]);
00074     if (lon0 < -360 && lon1 > 360) {
00075       lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00076       lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00077     }
00078     nx = ny = 0;
00079     for (lon = lon0; lon <= lon1; lon += dlon) {
00080       lons[nx] = lon;
00081       if ((++nx) > EX)
00082         ERRMSG("Too many longitudes!");
00083     }
00084     if (lat0 < -90 && lat1 > 90) {
00085       lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
```

```
00086       lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00087     }
00088     for (lat = lat0; lat <= lat1; lat += dlat) {
00089       lats[ny] = lat;
00090       if ((++ny) > EY)
00091         ERRMSG("Too many latitudes!");
00092     }
00093
00094     /* Average data... */
00095     for (ix = 0; ix < nx; ix++)
00096       for (iy = 0; iy < ny; iy++) {
00097
00098         /* Interpolate to given log-pressure height... */
00099         intpol_met_space(met, p0, lons[ix], lats[iy], &ps, &pt,
00100                          &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00101
00102         /* Get tropopause data... */
00103         intpol_met_space(met, pt, lons[ix], lats[iy], NULL, NULL,
00104                          &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00105
00106         /* Sum up data... */
00107         timem[ix][iy] += met->time;
00108         zm[ix][iy] += z;
00109         tm[ix][iy] += t;
00110         um[ix][iy] += u;
00111         vm[ix][iy] += v;
00112         wm[ix][iy] += w;
00113         pvm[ix][iy] += pv;
00114         h2om[ix][iy] += h2o;
00115         o3m[ix][iy] += o3;
00116         psm[ix][iy] += ps;
00117         ptm[ix][iy] += pt;
00118         ztm[ix][iy] += zt;
00119         ttm[ix][iy] += tt;
00120         np[ix][iy]++;
00121       }
00122   }
00123
00124   /* Create output file... */
00125   printf("Write meteorological data file: %s\n", argv[2]);
00126   if (!(out = fopen(argv[2], "w")))
00127     ERRMSG("Cannot create file!");
00128
00129   /* Write header... */
00130   fprintf(out,
00131           "# $1  = time [s]\n"
00132           "# $2  = altitude [km]\n"
00133           "# $3  = longitude [deg]\n"
00134           "# $4  = latitude [deg]\n"
00135           "# $5  = pressure [hPa]\n"
00136           "# $6  = temperature [K]\n"
00137           "# $7  = zonal wind [m/s]\n"
00138           "# $8  = meridional wind [m/s]\n"
00139           "# $9  = vertical wind [hPa/s]\n"
00140           "# $10 = H2O volume mixing ratio [1]\n");
00141   fprintf(out,
00142           "# $11 = O3 volume mixing ratio [1]\n"
00143           "# $12 = geopotential height [km]\n"
00144           "# $13 = potential vorticity [PVU]\n"
00145           "# $14 = surface pressure [hPa]\n"
00146           "# $15 = tropopause pressure [hPa]\n"
00147           "# $16 = tropopause geopotential height [km]\n"
00148           "# $17 = tropopause temperature [K]\n");
00149
00150   /* Write data... */
00151   for (iy = 0; iy < ny; iy++) {
00152     fprintf(out, "\n");
00153     for (ix = 0; ix < nx; ix++)
00154       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00155               timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
00156               tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00157               vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00158               h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00159               zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00160               psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00161               ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00162   }
00163
00164   /* Close file... */
00165   fclose(out);
00166
00167   /* Free... */
00168   free(met);
00169
00170   return EXIT_SUCCESS;
00171 }
```

Here is the call graph for this function:



5.24 met_map.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *out;
00036
00037   static double timem[EX][EY], p0, ps, psm[EX][EY], pt, ptm[EX][EY], t,
00038     tm[EX][EY], u, um[EX][EY], v, vm[EX][EY], w, wm[EX][EY], h2o,
00039     h2om[EX][EY], o3, o3m[EX][EY], z, zm[EX][EY], pv, pvm[EX][EY], zt,
00040     ztm[EX][EY], tt, ttm[EX][EY], lon, lon0, lon1, lons[EX], dlon, lat, lat0,
00041     lat1, lats[EY], dlat;
00042
00043   static int i, ix, iy, np[EX][EY], nx, ny;
```

```
00044
00045   /* Allocate... */
00046   ALLOC(met, met_t, 1);
00047
00048   /* Check arguments... */
00049   if (argc < 4)
00050     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00051
00052   /* Read control parameters... */
00053   read_ctl(argv[1], argc, argv, &ctl);
00054   p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "", NULL));
00055   lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00056   lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00057   dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00058   lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00059   lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00060   dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00061
00062   /* Loop over files... */
00063   for (i = 3; i < argc; i++) {
00064
00065     /* Read meteorological data... */
00066     if (!read_met(&ctl, argv[i], met))
00067       continue;
00068
00069     /* Set horizontal grid... */
00070     if (dlon <= 0)
00071       dlon = fabs(met->lon[1] - met->lon[0]);
00072     if (dlat <= 0)
00073       dlat = fabs(met->lat[1] - met->lat[0]);
00074     if (lon0 < -360 && lon1 > 360) {
00075       lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00076       lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00077     }
00078     nx = ny = 0;
00079     for (lon = lon0; lon <= lon1; lon += dlon) {
00080       lons[nx] = lon;
00081       if ((++nx) > EX)
00082         ERRMSG("Too many longitudes!");
00083     }
00084     if (lat0 < -90 && lat1 > 90) {
00085       lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00086       lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00087     }
00088     for (lat = lat0; lat <= lat1; lat += dlat) {
00089       lats[ny] = lat;
00090       if ((++ny) > EY)
00091         ERRMSG("Too many latitudes!");
00092     }
00093
00094     /* Average data... */
00095     for (ix = 0; ix < nx; ix++)
00096       for (iy = 0; iy < ny; iy++) {
00097
00098         /* Interpolate to given log-pressure height... */
00099         intpol_met_space(met, p0, lons[ix], lats[iy], &ps, &pt,
00100                          &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00101
00102         /* Get tropopause data... */
00103         intpol_met_space(met, pt, lons[ix], lats[iy], NULL, NULL,
00104                          &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00105
00106         /* Sum up data... */
00107         timem[ix][iy] += met->time;
00108         zm[ix][iy] += z;
00109         tm[ix][iy] += t;
00110         um[ix][iy] += u;
00111         vm[ix][iy] += v;
00112         wm[ix][iy] += w;
00113         pvm[ix][iy] += pv;
00114         h2om[ix][iy] += h2o;
00115         o3m[ix][iy] += o3;
00116         psm[ix][iy] += ps;
00117         ptm[ix][iy] += pt;
00118         ztm[ix][iy] += zt;
00119         ttm[ix][iy] += tt;
00120         np[ix][iy]++;
00121       }
00122   }
00123
00124   /* Create output file... */
00125   printf("Write meteorological data file: %s\n", argv[2]);
00126   if (!(out = fopen(argv[2], "w")))
00127     ERRMSG("Cannot create file!");
00128
00129   /* Write header... */
00130   fprintf(out,
```

```
00131           "# $1  = time [s]\n"
00132           "# $2  = altitude [km]\n"
00133           "# $3  = longitude [deg]\n"
00134           "# $4  = latitude [deg]\n"
00135           "# $5  = pressure [hPa]\n"
00136           "# $6  = temperature [K]\n"
00137           "# $7  = zonal wind [m/s]\n"
00138           "# $8  = meridional wind [m/s]\n"
00139           "# $9  = vertical wind [hPa/s]\n"
00140           "# $10 = H2O volume mixing ratio [1]\n");
00141   fprintf(out,
00142           "# $11 = O3 volume mixing ratio [1]\n"
00143           "# $12 = geopotential height [km]\n"
00144           "# $13 = potential vorticity [PVU]\n"
00145           "# $14 = surface pressure [hPa]\n"
00146           "# $15 = tropopause pressure [hPa]\n"
00147           "# $16 = tropopause geopotential height [km]\n"
00148           "# $17 = tropopause temperature [K]\n");
00149
00150   /* Write data... */
00151   for (iy = 0; iy < ny; iy++) {
00152     fprintf(out, "\n");
00153     for (ix = 0; ix < nx; ix++)
00154       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00155               timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
00156               tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00157               vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00158               h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00159               zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00160               psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00161               ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
00162   }
00163
00164   /* Close file... */
00165   fclose(out);
00166
00167   /* Free... */
00168   free(met);
00169
00170   return EXIT_SUCCESS;
00171 }
```

5.25 met_prof.c File Reference

Extract vertical profile from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.25.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met_prof.c.

5.25.2 Function Documentation

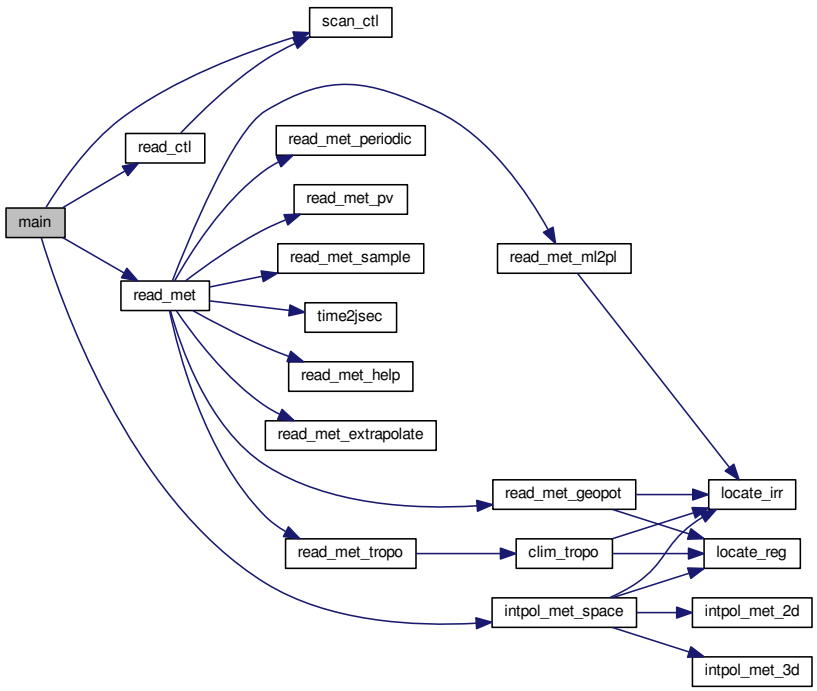
5.25.2.1 int main (int *argc*, char ∗ *argv[ ]*)

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], pv, pvm[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, "PROF_Z0", -1, "0", NULL);
00065   z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "60", NULL);
00066   dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "1", NULL);
00067   lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "", NULL);
00068   lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "", NULL);
00069   dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00070   lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "", NULL);
00071   lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "", NULL);
00072   dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00073
00074   /* Loop over input files... */
00075   for (i = 3; i < argc; i++) {
00076
00077     /* Read meteorological data... */
00078     if (!read_met(&ctl, argv[i], met))
00079       continue;
00080
00081     /* Set horizontal grid spacing... */
00082     if (dlon <= 0)
00083       dlon = fabs(met->lon[1] - met->lon[0]);
00084     if (dlat <= 0)
00085       dlat = fabs(met->lat[1] - met->lat[0]);
00086
00087     /* Average... */
00088     for (z = z0; z <= z1; z += dz) {
00089       iz = (int) ((z - z0) / dz);
00090       if (iz < 0 || iz > NZ)
00091         ERRMSG("Too many altitudes!");
00092       for (lon = lon0; lon <= lon1; lon += dlon)
00093         for (lat = lat0; lat <= lat1; lat += dlat) {
00094           intpol_met_space(met, P(z), lon, lat, &ps, &pt, &zg,
00095                            &t, &u, &v, &w, &pv, &h2o, &o3);
00096           intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
00097                            &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00098           if (gsl_finite(t) && gsl_finite(u)
00099               && gsl_finite(v) && gsl_finite(w)) {
00100             timem[iz] += met->time;
00101             lonm[iz] += lon;
00102             latm[iz] += lat;
00103             zgm[iz] += zg;
00104             tm[iz] += t;
00105             um[iz] += u;
00106             vm[iz] += v;
00107             wm[iz] += w;
00108             pvm[iz] += pv;
00109             h2om[iz] += h2o;
00110             o3m[iz] += o3;
00111             psm[iz] += ps;
00112             ptm[iz] += pt;
00113             ztm[iz] += zt;
00114             ttm[iz] += tt;
00115             np[iz]++;
00116           }
00117         }
00118     }
00119   }
00120
00121   /* Create output file... */
00122   printf("Write meteorological data file: %s\n", argv[2]);
00123   if (!(out = fopen(argv[2], "w")))
00124     ERRMSG("Cannot create file!");
00125
00126   /* Write header... */
```

```
00127   fprintf(out,
00128           "# $1  = time [s]\n"
00129           "# $2  = altitude [km]\n"
00130           "# $3  = longitude [deg]\n"
00131           "# $4  = latitude [deg]\n"
00132           "# $5  = pressure [hPa]\n"
00133           "# $6  = temperature [K]\n"
00134           "# $7  = zonal wind [m/s]\n"
00135           "# $8  = meridional wind [m/s]\n"
00136           "# $9  = vertical wind [hPa/s]\n");
00137   fprintf(out,
00138           "# $10 = H2O volume mixing ratio [1]\n"
00139           "# $11 = O3 volume mixing ratio [1]\n"
00140           "# $12 = geopotential height [km]\n"
00141           "# $13 = potential vorticity [PVU]\n"
00142           "# $14 = surface pressure [hPa]\n"
00143           "# $15 = tropopause pressure [hPa]\n"
00144           "# $16 = tropopause geopotential height [km]\n"
00145           "# $17 = tropopause temperature [K]\n\n");
00146
00147   /* Write data... */
00148   for (z = z0; z <= z1; z += dz) {
00149     iz = (int) ((z - z0) / dz);
00150     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00151             timem[iz] / np[iz], z, lonm[iz] / np[iz], latm[iz] / np[iz], P(z),
00152             tm[iz] / np[iz], um[iz] / np[iz], vm[iz] / np[iz],
00153             wm[iz] / np[iz], h2om[iz] / np[iz], o3m[iz] / np[iz],
00154             zgm[iz] / np[iz], pvm[iz] / np[iz], psm[iz] / np[iz],
00155             ptm[iz] / np[iz], ztm[iz] / np[iz], ttm[iz] / np[iz]);
00156   }
00157
00158   /* Close file... */
00159   fclose(out);
00160
00161   /* Free... */
00162   free(met);
00163
00164   return EXIT_SUCCESS;
00165 }
```

Here is the call graph for this function:



5.26 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) 2013-2019 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], pv, pvm[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, "PROF_Z0", -1, "0", NULL);
00065   z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "60", NULL);
00066   dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "1", NULL);
00067   lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "", NULL);
00068   lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "", NULL);
00069   dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00070   lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "", NULL);
00071   lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "", NULL);
00072   dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00073
00074   /* Loop over input files... */
00075   for (i = 3; i < argc; i++) {
00076
00077     /* Read meteorological data... */
00078     if (!read_met(&ctl, argv[i], met))
00079       continue;
00080
00081     /* Set horizontal grid spacing... */
00082     if (dlon <= 0)
00083       dlon = fabs(met->lon[1] - met->lon[0]);
00084     if (dlat <= 0)
00085       dlat = fabs(met->lat[1] - met->lat[0]);
00086
00087     /* Average... */
00088     for (z = z0; z <= z1; z += dz) {
00089       iz = (int) ((z - z0) / dz);
```

```
00090       if (iz < 0 || iz > NZ)
00091         ERRMSG("Too many altitudes!");
00092       for (lon = lon0; lon <= lon1; lon += dlon)
00093         for (lat = lat0; lat <= lat1; lat += dlat) {
00094           intpol_met_space(met, P(z), lon, lat, &ps, &pt, &zg,
00095                            &t, &u, &v, &w, &pv, &h2o, &o3);
00096           intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
00097                            &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00098           if (gsl_finite(t) && gsl_finite(u)
00099               && gsl_finite(v) && gsl_finite(w)) {
00100             timem[iz] += met->time;
00101             lonm[iz] += lon;
00102             latm[iz] += lat;
00103             zgm[iz] += zg;
00104             tm[iz] += t;
00105             um[iz] += u;
00106             vm[iz] += v;
00107             wm[iz] += w;
00108             pvm[iz] += pv;
00109             h2om[iz] += h2o;
00110             o3m[iz] += o3;
00111             psm[iz] += ps;
00112             ptm[iz] += pt;
00113             ztm[iz] += zt;
00114             ttm[iz] += tt;
00115             np[iz]++;
00116           }
00117         }
00118     }
00119   }
00120
00121   /* Create output file... */
00122   printf("Write meteorological data file: %s\n", argv[2]);
00123   if (!(out = fopen(argv[2], "w")))
00124     ERRMSG("Cannot create file!");
00125
00126   /* Write header... */
00127   fprintf(out,
00128           "# $1  = time [s]\n"
00129           "# $2  = altitude [km]\n"
00130           "# $3  = longitude [deg]\n"
00131           "# $4  = latitude [deg]\n"
00132           "# $5  = pressure [hPa]\n"
00133           "# $6  = temperature [K]\n"
00134           "# $7  = zonal wind [m/s]\n"
00135           "# $8  = meridional wind [m/s]\n"
00136           "# $9  = vertical wind [hPa/s]\n");
00137   fprintf(out,
00138           "# $10 = H2O volume mixing ratio [1]\n"
00139           "# $11 = O3 volume mixing ratio [1]\n"
00140           "# $12 = geopotential height [km]\n"
00141           "# $13 = potential vorticity [PVU]\n"
00142           "# $14 = surface pressure [hPa]\n"
00143           "# $15 = tropopause pressure [hPa]\n"
00144           "# $16 = tropopause geopotential height [km]\n"
00145           "# $17 = tropopause temperature [K]\n\n");
00146
00147   /* Write data... */
00148   for (z = z0; z <= z1; z += dz) {
00149     iz = (int) ((z - z0) / dz);
00150     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00151             timem[iz] / np[iz], z, lonm[iz] / np[iz], latm[iz] / np[iz], P(z),
00152             tm[iz] / np[iz], um[iz] / np[iz], vm[iz] / np[iz],
00153             wm[iz] / np[iz], h2om[iz] / np[iz], o3m[iz] / np[iz],
00154             zgm[iz] / np[iz], pvm[iz] / np[iz], psm[iz] / np[iz],
00155             ptm[iz] / np[iz], ztm[iz] / np[iz], ttm[iz] / np[iz]);
00156   }
00157
00158   /* Close file... */
00159   fclose(out);
00160
00161   /* Free... */
00162   free(met);
00163
00164   return EXIT_SUCCESS;
00165 }
```

5.27 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

- int main (int argc, char ∗argv[ ])

5.27.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

5.27.2 Function Documentation

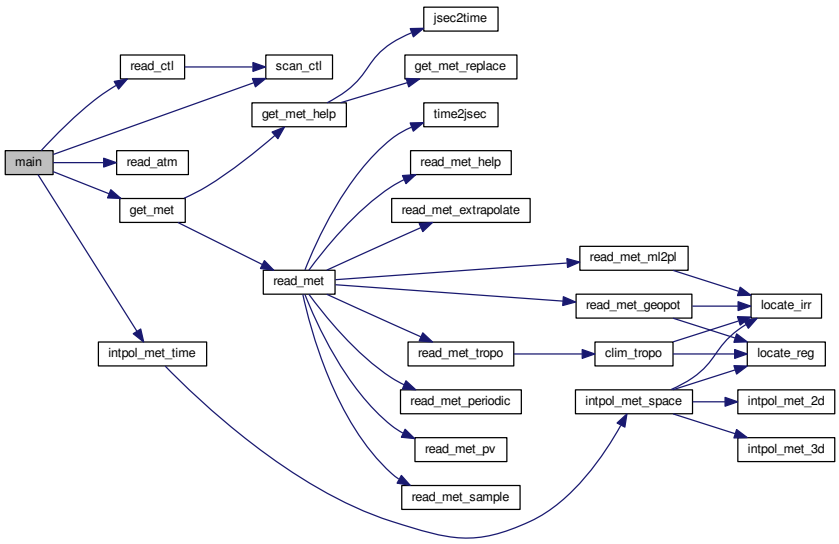
5.27.2.1 int main (int *argc,* char ∗ *argv[ ]*)

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, p0, p1, pref, ps, pt, pv, t, tt, u, v, w, z, zm, zref, zt;
00044
00045   int geopot, ip, it;
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");
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   geopot =
00059     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00060
00061   /* Read atmospheric data... */
00062   if (!read_atm(argv[4], &ctl, atm))
00063     ERRMSG("Cannot open file!");
00064
00065   /* Create output file... */
00066   printf("Write meteorological data file: %s\n", argv[2]);
00067   if (!(out = fopen(argv[2], "w")))
00068     ERRMSG("Cannot create file!");
00069
00070   /* Write header... */
00071   fprintf(out,
00072           "# $1  = time [s]\n"
00073           "# $2  = altitude [km]\n"
00074           "# $3  = longitude [deg]\n"
00075           "# $4  = latitude [deg]\n"
00076           "# $5  = pressure [hPa]\n"
00077           "# $6  = temperature [K]\n"
00078           "# $7  = zonal wind [m/s]\n"
00079           "# $8  = meridional wind [m/s]\n"
00080           "# $9  = vertical wind [hPa/s]\n");
00081   fprintf(out,
00082           "# $10 = H2O volume mixing ratio [1]\n"
00083           "# $11 = O3 volume mixing ratio [1]\n"
00084           "# $12 = geopotential height [km]\n"
00085           "# $13 = potential vorticity [PVU]\n"
00086           "# $14 = surface pressure [hPa]\n"
00087           "# $15 = tropopause pressure [hPa]\n"
00088           "# $16 = tropopause geopotential height [km]\n"
00089           "# $17 = tropopause temperature [K]\n\n");
```

```
00090
00091   /* Loop over air parcels... */
00092   for (ip = 0; ip < atm->np; ip++) {
00093
00094     /* Get meteorological data... */
00095     get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);
00096
00097     /* Set reference pressure for interpolation... */
00098     pref = atm->p[ip];
00099     if (geopot) {
00100       zref = Z(pref);
00101       p0 = met0->p[0];
00102       p1 = met0->p[met0->np - 1];
00103       for (it = 0; it < 24; it++) {
00104         pref = 0.5 * (p0 + p1);
00105         intpol_met_time(met0, met1, atm->time[ip], pref, atm->
      lon[ip],
00106                         atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
00107                         NULL, NULL, NULL);
00108         if (zref > zm || !gsl_finite(zm))
00109           p0 = pref;
00110         else
00111           p1 = pref;
00112       }
00113       pref = 0.5 * (p0 + p1);
00114     }
00115
00116     /* Interpolate meteorological data... */
00117     intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
00118                     atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00119                     &o3);
00120     intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
      lat[ip],
00121                     NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00122
00123     /* Write data... */
00124     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00125             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00126             atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
00127   }
00128
00129   /* Close file... */
00130   fclose(out);
00131
00132   /* Free... */
00133   free(atm);
00134   free(met0);
00135   free(met1);
00136
00137   return EXIT_SUCCESS;
00138 }
```

Here is the call graph for this function:



5.28 met_sample.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 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, p0, p1, pref, ps, pt, pv, t, tt, u, v, w, z, zm, zref, zt;
00044
00045   int geopot, ip, it;
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");
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   geopot =
00059     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00060
00061   /* Read atmospheric data... */
00062   if (!read_atm(argv[4], &ctl, atm))
00063     ERRMSG("Cannot open file!");
00064
00065   /* Create output file... */
00066   printf("Write meteorological data file: %s\n", argv[2]);
00067   if (!(out = fopen(argv[2], "w")))
00068     ERRMSG("Cannot create file!");
00069
00070   /* Write header... */
00071   fprintf(out,
00072           "# $1  = time [s]\n"
00073           "# $2  = altitude [km]\n"
00074           "# $3  = longitude [deg]\n"
00075           "# $4  = latitude [deg]\n"
00076           "# $5  = pressure [hPa]\n"
00077           "# $6  = temperature [K]\n"
00078           "# $7  = zonal wind [m/s]\n"
00079           "# $8  = meridional wind [m/s]\n"
00080           "# $9  = vertical wind [hPa/s]\n");
00081   fprintf(out,
00082           "# $10 = H2O volume mixing ratio [1]\n"
00083           "# $11 = O3 volume mixing ratio [1]\n"
00084           "# $12 = geopotential height [km]\n"
00085           "# $13 = potential vorticity [PVU]\n"
00086           "# $14 = surface pressure [hPa]\n"
00087           "# $15 = tropopause pressure [hPa]\n"
00088           "# $16 = tropopause geopotential height [km]\n"
00089           "# $17 = tropopause temperature [K]\n\n");
```

```
00090
00091   /* Loop over air parcels... */
00092   for (ip = 0; ip < atm->np; ip++) {
00093
00094     /* Get meteorological data... */
00095     get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);
00096
00097     /* Set reference pressure for interpolation... */
00098     pref = atm->p[ip];
00099     if (geopot) {
00100       zref = Z(pref);
00101       p0 = met0->p[0];
00102       p1 = met0->p[met0->np - 1];
00103       for (it = 0; it < 24; it++) {
00104         pref = 0.5 * (p0 + p1);
00105         intpol_met_time(met0, met1, atm->time[ip], pref, atm->
      lon[ip],
00106                         atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
00107                         NULL, NULL, NULL);
00108         if (zref > zm || !gsl_finite(zm))
00109           p0 = pref;
00110         else
00111           p1 = pref;
00112       }
00113       pref = 0.5 * (p0 + p1);
00114     }
00115
00116     /* Interpolate meteorological data... */
00117     intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
00118                     atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
00119                     &o3);
00120     intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
      lat[ip],
00121                     NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);
00122
00123     /* Write data... */
00124     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00125             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00126             atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
00127   }
00128
00129   /* Close file... */
00130   fclose(out);
00131
00132   /* Free... */
00133   free(atm);
00134   free(met0);
00135   free(met1);
00136
00137   return EXIT_SUCCESS;
00138 }
```

5.29 met_zm.c File Reference

Extract zonal mean from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.29.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met_zm.c.

5.29.2 Function Documentation

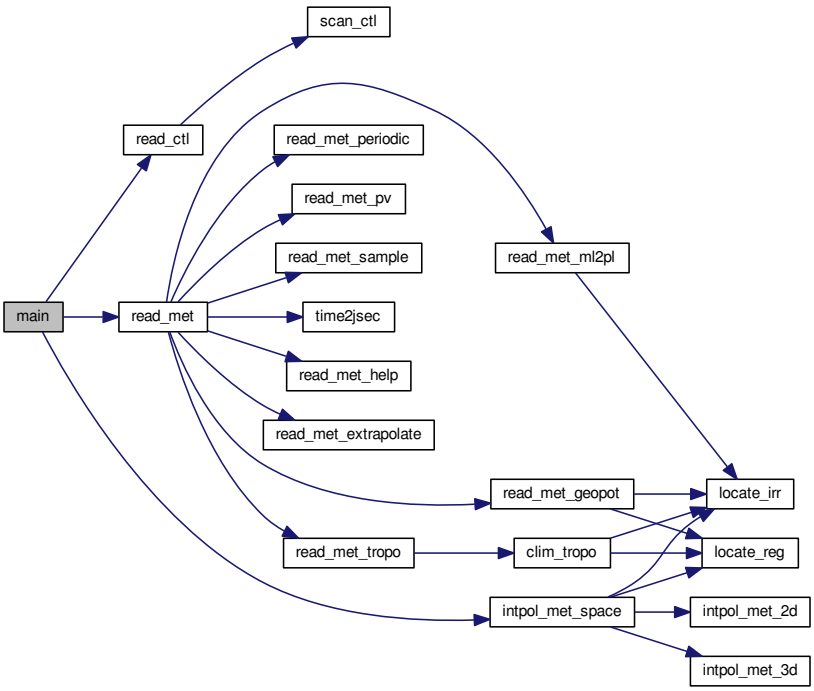
5.29.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file met_zm.c.

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

```
00104   fprintf(out,
00105           "# $11 = O3 volume mixing ratio [1]\n"
00106           "# $12 = geopotential height [km]\n"
00107           "# $13 = potential vorticity [PVU]\n"
00108           "# $14 = surface pressure [hPa]\n"
00109           "# $15 = tropopause pressure [hPa]\n"
00110           "# $16 = tropopause geopotential height [km]\n"
00111           "# $17 = tropopause temperature [K]\n");
00112
00113   /* Write data... */
00114   for (ip = 0; ip < met->np; ip++) {
00115     fprintf(out, "\n");
00116     for (iy = 0; iy < met->ny; iy++)
00117       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00118               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), 0.0, met->lat[iy],
00119               met->p[ip], tm[ip][iy] / np[ip][iy], um[ip][iy] / np[ip][iy],
00120               vm[ip][iy] / np[ip][iy], wm[ip][iy] / np[ip][iy],
00121               h2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
00122               zm[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
00123               psm[ip][iy] / np[ip][iy], ptm[ip][iy] / npt[ip][iy],
00124               ztm[ip][iy] / npt[ip][iy], ttm[ip][iy] / npt[ip][iy]);
00125   }
00126
00127   /* Close file... */
00128   fclose(out);
00129
00130   /* Free... */
00131   free(met);
00132
00133   return EXIT_SUCCESS;
00134 }
```

Here is the call graph for this function:



5.30 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   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *out;
00036
00037   static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], ttm[EP][EY],
00038     ztm[EP][EY], tm[EP][EY], um[EP][EY], vm[EP][EY], wm[EP][EY], h2om[EP][EY],
00039     pvm[EP][EY], o3m[EP][EY], zm[EP][EY], zt, tt;
00040
00041   static int i, ip, ix, iy, np[EP][EY], npt[EP][EY];
00042
00043   /* Allocate... */
00044   ALLOC(met, met_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052
00053   /* Loop over files... */
00054   for (i = 3; i < argc; i++) {
00055
00056     /* Read meteorological data... */
00057     if (!read_met(&ctl, argv[i], met))
00058       continue;
00059
00060     /* Average data... */
00061     for (ix = 0; ix < met->nx; ix++)
00062       for (iy = 0; iy < met->ny; iy++)
00063         for (ip = 0; ip < met->np; ip++) {
00064           intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
      lat[iy],
00065                            NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
00066                            NULL);
00067           timem[ip][iy] += met->time;
00068           zm[ip][iy] += met->z[ix][iy][ip];
00069           tm[ip][iy] += met->t[ix][iy][ip];
00070           um[ip][iy] += met->u[ix][iy][ip];
00071           vm[ip][iy] += met->v[ix][iy][ip];
00072           wm[ip][iy] += met->w[ix][iy][ip];
00073           pvm[ip][iy] += met->pv[ix][iy][ip];
00074           h2om[ip][iy] += met->h2o[ix][iy][ip];
00075           o3m[ip][iy] += met->o3[ix][iy][ip];
00076           psm[ip][iy] += met->ps[ix][iy];
00077           if (gsl_finite(met->pt[ix][iy])) {
00078             ptm[ip][iy] += met->pt[ix][iy];
00079             ztm[ip][iy] += zt;
00080             ttm[ip][iy] += tt;
00081             npt[ip][iy]++;
00082           }
00083           np[ip][iy]++;
00084         }
00085   }
00086
00087   /* Create output file... */
00088   printf("Write meteorological data file: %s\n", argv[2]);
00089   if (!(out = fopen(argv[2], "w")))
00090     ERRMSG("Cannot create file!");
00091
00092   /* Write header... */
00093   fprintf(out,
00094           "# $1  = time [s]\n"
00095           "# $2  = altitude [km]\n"
```

```
00096           "# $3  = longitude [deg]\n"
00097           "# $4  = latitude [deg]\n"
00098           "# $5  = pressure [hPa]\n"
00099           "# $6  = temperature [K]\n"
00100           "# $7  = zonal wind [m/s]\n"
00101           "# $8  = meridional wind [m/s]\n"
00102           "# $9  = vertical wind [hPa/s]\n"
00103           "# $10 = H2O volume mixing ratio [1]\n");
00104   fprintf(out,
00105           "# $11 = O3 volume mixing ratio [1]\n"
00106           "# $12 = geopotential height [km]\n"
00107           "# $13 = potential vorticity [PVU]\n"
00108           "# $14 = surface pressure [hPa]\n"
00109           "# $15 = tropopause pressure [hPa]\n"
00110           "# $16 = tropopause geopotential height [km]\n"
00111           "# $17 = tropopause temperature [K]\n");
00112
00113   /* Write data... */
00114   for (ip = 0; ip < met->np; ip++) {
00115     fprintf(out, "\n");
00116     for (iy = 0; iy < met->ny; iy++)
00117       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00118               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), 0.0, met->lat[iy],
00119               met->p[ip], tm[ip][iy] / np[ip][iy], um[ip][iy] / np[ip][iy],
00120               vm[ip][iy] / np[ip][iy], wm[ip][iy] / np[ip][iy],
00121               h2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
00122               zm[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
00123               psm[ip][iy] / np[ip][iy], ptm[ip][iy] / npt[ip][iy],
00124               ztm[ip][iy] / npt[ip][iy], ttm[ip][iy] / npt[ip][iy]);
00125   }
00126
00127   /* Close file... */
00128   fclose(out);
00129
00130   /* Free... */
00131   free(met);
00132
00133   return EXIT_SUCCESS;
00134 }
```

5.31 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

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

5.31.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

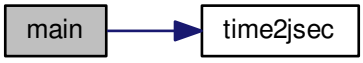
5.31.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

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

Here is the call graph for this function:



5.32 time2jsec.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 8)
00037     ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039   /* Read arguments... */
00040   year = atoi(argv[1]);
```

```
00041   mon = atoi(argv[2]);
00042   day = atoi(argv[3]);
00043   hour = atoi(argv[4]);
00044   min = atoi(argv[5]);
00045   sec = atoi(argv[6]);
00046   remain = atof(argv[7]);
00047
00048   /* Convert... */
00049   time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
00050   printf("%.2f\n", jsec);
00051
00052   return EXIT_SUCCESS;
00053 }
```

5.33 trac.c File Reference

Lagrangian particle dispersion model.

Functions

- void 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[])

5.33.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.33.2 Function Documentation

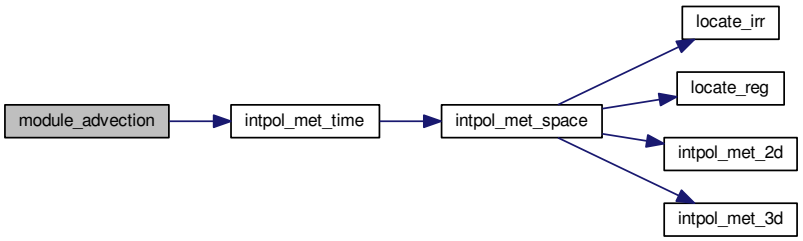
5.33.2.1 void module_advection (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate advection of air parcels.

Definition at line 387 of file trac.c.

```
00392             {
00393
00394   double v[3], xm[3];
00395
00396   /* Interpolate meteorological data... */
00397   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00398                   atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00399                   &v[0], &v[1], &v[2], NULL, NULL, NULL);
00400
00401   /* Get position of the mid point... */
00402   xm[0] = atm->lon[ip] + DX2DEG(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00403   xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt * v[1] / 1000.);
00404   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00405
00406   /* Interpolate meteorological data for mid point... */
00407   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00408                   xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
00409                   &v[0], &v[1], &v[2], NULL, NULL, NULL);
00410
00411   /* Save new position... */
00412   atm->time[ip] += dt;
00413   atm->lon[ip] += DX2DEG(dt * v[0] / 1000., xm[1]);
00414   atm->lat[ip] += DY2DEG(dt * v[1] / 1000.);
00415   atm->p[ip] += dt * v[2];
00416 }
```

Here is the call graph for this function:



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

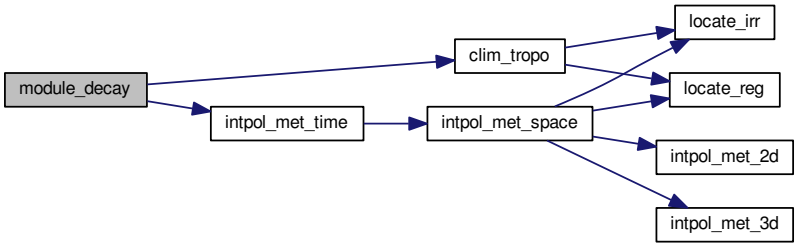
Calculate exponential decay of particle mass.

Definition at line 420 of file trac.c.

```
00426             {
00427
00428   double ps, pt, tdec;
00429
00430   /* Set constant lifetime... */
00431   if (ctl->tdec_trop == ctl->tdec_strat)
00432     tdec = ctl->tdec_trop;
00433
00434   /* Set altitude-dependent lifetime... */
00435   else {
```

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

Here is the call graph for this function:



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

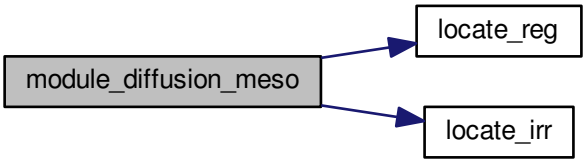
Calculate mesoscale diffusion.

Definition at line 458 of file trac.c.

```
00465                 {
00466
00467   double r, rs, u[16], v[16], w[16];
00468
00469   int ix, iy, iz;
00470
00471   /* Get indices... */
00472   ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00473   iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00474   iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00475
00476   /* Caching of wind standard deviations... */
00477   if (atm->cache_time[ix][iy][iz] != met0->time) {
00478
00479     /* Collect local wind data... */
00480     u[0] = met0->u[ix][iy][iz];
00481     u[1] = met0->u[ix + 1][iy][iz];
00482     u[2] = met0->u[ix][iy + 1][iz];
00483     u[3] = met0->u[ix + 1][iy + 1][iz];
00484     u[4] = met0->u[ix][iy][iz + 1];
00485     u[5] = met0->u[ix + 1][iy][iz + 1];
00486     u[6] = met0->u[ix][iy + 1][iz + 1];
00487     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00488
00489     v[0] = met0->v[ix][iy][iz];
00490     v[1] = met0->v[ix + 1][iy][iz];
00491     v[2] = met0->v[ix][iy + 1][iz];
```

```
00492     v[3] = met0->v[ix + 1][iy + 1][iz];
00493     v[4] = met0->v[ix][iy][iz + 1];
00494     v[5] = met0->v[ix + 1][iy][iz + 1];
00495     v[6] = met0->v[ix][iy + 1][iz + 1];
00496     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00497
00498     w[0] = met0->w[ix][iy][iz];
00499     w[1] = met0->w[ix + 1][iy][iz];
00500     w[2] = met0->w[ix][iy + 1][iz];
00501     w[3] = met0->w[ix + 1][iy + 1][iz];
00502     w[4] = met0->w[ix][iy][iz + 1];
00503     w[5] = met0->w[ix + 1][iy][iz + 1];
00504     w[6] = met0->w[ix][iy + 1][iz + 1];
00505     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00506
00507     /* Collect local wind data... */
00508     u[8] = met1->u[ix][iy][iz];
00509     u[9] = met1->u[ix + 1][iy][iz];
00510     u[10] = met1->u[ix][iy + 1][iz];
00511     u[11] = met1->u[ix + 1][iy + 1][iz];
00512     u[12] = met1->u[ix][iy][iz + 1];
00513     u[13] = met1->u[ix + 1][iy][iz + 1];
00514     u[14] = met1->u[ix][iy + 1][iz + 1];
00515     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00516
00517     v[8] = met1->v[ix][iy][iz];
00518     v[9] = met1->v[ix + 1][iy][iz];
00519     v[10] = met1->v[ix][iy + 1][iz];
00520     v[11] = met1->v[ix + 1][iy + 1][iz];
00521     v[12] = met1->v[ix][iy][iz + 1];
00522     v[13] = met1->v[ix + 1][iy][iz + 1];
00523     v[14] = met1->v[ix][iy + 1][iz + 1];
00524     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00525
00526     w[8] = met1->w[ix][iy][iz];
00527     w[9] = met1->w[ix + 1][iy][iz];
00528     w[10] = met1->w[ix][iy + 1][iz];
00529     w[11] = met1->w[ix + 1][iy + 1][iz];
00530     w[12] = met1->w[ix][iy][iz + 1];
00531     w[13] = met1->w[ix + 1][iy][iz + 1];
00532     w[14] = met1->w[ix][iy + 1][iz + 1];
00533     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00534
00535     /* Get standard deviations of local wind data... */
00536     atm->cache_usig[ix][iy][iz] = (float) gsl_stats_sd(u, 1, 16);
00537     atm->cache_vsig[ix][iy][iz] = (float) gsl_stats_sd(v, 1, 16);
00538     atm->cache_wsig[ix][iy][iz] = (float) gsl_stats_sd(w, 1, 16);
00539     atm->cache_time[ix][iy][iz] = met0->time;
00540   }
00541
00542   /* Set temporal correlations for mesoscale fluctuations... */
00543   r = 1 - 2 * fabs(dt) / ctl->dt_met;
00544   rs = sqrt(1 - r * r);
00545
00546   /* Calculate horizontal mesoscale wind fluctuations... */
00547   if (ctl->turb_mesox > 0) {
00548     atm->up[ip] = (float)
00549       (r * atm->up[ip]
00550        + rs * gsl_ran_gaussian_ziggurat(rng,
00551                                         ctl->turb_mesox *
00552                                         atm->cache_usig[ix][iy][iz]));
00553     atm->lon[ip] += DX2DEG(atm->up[ip] * dt / 1000., atm->lat[ip]);
00554
00555     atm->vp[ip] = (float)
00556       (r * atm->vp[ip]
00557        + rs * gsl_ran_gaussian_ziggurat(rng,
00558                                         ctl->turb_mesox *
00559                                         atm->cache_vsig[ix][iy][iz]));
00560     atm->lat[ip] += DY2DEG(atm->vp[ip] * dt / 1000.);
00561   }
00562
00563   /* Calculate vertical mesoscale wind fluctuations... */
00564   if (ctl->turb_mesoz > 0) {
00565     atm->wp[ip] = (float)
00566       (r * atm->wp[ip]
00567        + rs * gsl_ran_gaussian_ziggurat(rng,
00568                                         ctl->turb_mesoz *
00569                                         atm->cache_wsig[ix][iy][iz]));
00570     atm->p[ip] += atm->wp[ip] * dt;
00571   }
00572 }
```

Here is the call graph for this function:



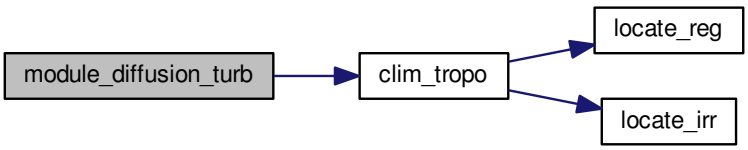
5.33.2.4 void module_diffusion_turb (ctl_t * *ctl*, atm_t * *atm*, int *ip*, double *dt*, gsl_rng * *rng*)

Calculate turbulent diffusion.

Definition at line 576 of file trac.c.

```
00581                 {
00582
00583   double dx, dz, pt, p0, p1, w;
00584
00585   /* Get tropopause pressure... */
00586   pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00587
00588   /* Get weighting factor... */
00589   p1 = pt * 0.866877899;
00590   p0 = pt / 0.866877899;
00591   if (atm->p[ip] > p0)
00592     w = 1;
00593   else if (atm->p[ip] < p1)
00594     w = 0;
00595   else
00596     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00597
00598   /* Set diffusivity... */
00599   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00600   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00601
00602   /* Horizontal turbulent diffusion... */
00603   if (dx > 0) {
00604     atm->lon[ip]
00605       += DX2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00606                 / 1000., atm->lat[ip]);
00607     atm->lat[ip]
00608       += DY2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00609                 / 1000.);
00610   }
00611
00612   /* Vertical turbulent diffusion... */
00613   if (dz > 0)
00614     atm->p[ip]
00615       += DZ2DP(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00616                / 1000., atm->p[ip]);
00617 }
```

Here is the call graph for this function:



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

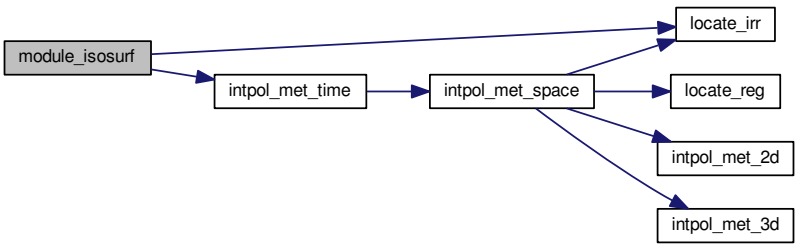
Force air parcels to stay on isosurface.

Definition at line 621 of file trac.c.

```
00626           {
00627
00628     static double *iso, *ps, t, *ts;
00629
00630     static int idx, ip2, n;
00631
00632     FILE *in;
00633
00634     char line[LEN];
00635
00636     /* Initialize... */
00637     if (ip < 0) {
00638
00639       /* Allocate... */
00640       ALLOC(iso, double,
00641             NP);
00642       ALLOC(ps, double,
00643             NP);
00644       ALLOC(ts, double,
00645             NP);
00646
00647       /* Save pressure... */
00648       if (ctl->isosurf == 1)
00649         for (ip2 = 0; ip2 < atm->np; ip2++)
00650           iso[ip2] = atm->p[ip2];
00651
00652       /* Save density... */
00653       else if (ctl->isosurf == 2)
00654         for (ip2 = 0; ip2 < atm->np; ip2++) {
00655           intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00656                           atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00657                           &t, NULL, NULL, NULL, NULL, NULL, NULL);
00658           iso[ip2] = atm->p[ip2] / t;
00659         }
00660
00661       /* Save potential temperature... */
00662       else if (ctl->isosurf == 3)
00663         for (ip2 = 0; ip2 < atm->np; ip2++) {
00664           intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00665                           atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00666                           &t, NULL, NULL, NULL, NULL, NULL, NULL);
00667           iso[ip2] = THETA(atm->p[ip2], t);
00668         }
00669
00670       /* Read balloon pressure data... */
00671       else if (ctl->isosurf == 4) {
00672
00673         /* Write info... */
00674         printf("Read balloon pressure data: %s\n", ctl->balloon);
00675
00676         /* Open file... */
00677         if (!(in = fopen(ctl->balloon, "r")))
00678           ERRMSG("Cannot open file!");
```

```
00679
00680       /* Read pressure time series... */
00681       while (fgets(line, LEN, in))
00682         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00683           if ((++n) > NP)
00684             ERRMSG("Too many data points!");
00685
00686       /* Check number of points... */
00687       if (n < 1)
00688         ERRMSG("Could not read any data!");
00689
00690       /* Close file... */
00691       fclose(in);
00692     }
00693
00694     /* Leave initialization... */
00695     return;
00696   }
00697
00698   /* Restore pressure... */
00699   if (ctl->isosurf == 1)
00700     atm->p[ip] = iso[ip];
00701
00702   /* Restore density... */
00703   else if (ctl->isosurf == 2) {
00704     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00705                     atm->lat[ip], NULL, NULL, NULL, &t,
00706                     NULL, NULL, NULL, NULL, NULL, NULL);
00707     atm->p[ip] = iso[ip] * t;
00708   }
00709
00710   /* Restore potential temperature... */
00711   else if (ctl->isosurf == 3) {
00712     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00713                     atm->lat[ip], NULL, NULL, NULL, &t,
00714                     NULL, NULL, NULL, NULL, NULL, NULL);
00715     atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00716   }
00717
00718   /* Interpolate pressure... */
00719   else if (ctl->isosurf == 4) {
00720     if (atm->time[ip] <= ts[0])
00721       atm->p[ip] = ps[0];
00722     else if (atm->time[ip] >= ts[n - 1])
00723       atm->p[ip] = ps[n - 1];
00724     else {
00725       idx = locate_irr(ts, n, atm->time[ip]);
00726       atm->p[ip] = LIN(ts[idx], ps[idx],
00727                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00728     }
00729   }
00730 }
```

Here is the call graph for this function:



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

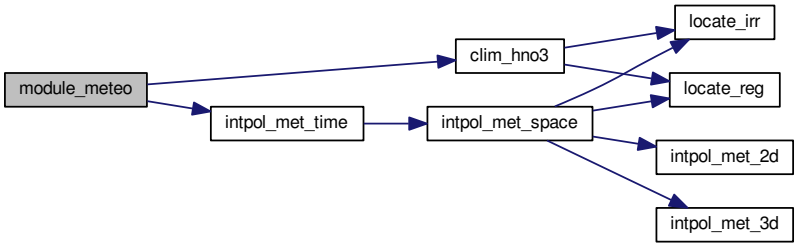
Interpolate meteorological data for air parcel positions.

Definition at line 734 of file trac.c.

```
00739           {
00740
00741   double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;
00742
00743   /* Interpolate meteorological data... */
00744   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00745                   atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00746
00747   /* Set surface pressure... */
00748   if (ctl->qnt_ps >= 0)
00749     atm->q[ctl->qnt_ps][ip] = ps;
00750
00751   /* Set tropopause pressure... */
00752   if (ctl->qnt_pt >= 0)
00753     atm->q[ctl->qnt_pt][ip] = pt;
00754
00755   /* Set pressure... */
00756   if (ctl->qnt_p >= 0)
00757     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00758
00759   /* Set geopotential height... */
00760   if (ctl->qnt_z >= 0)
00761     atm->q[ctl->qnt_z][ip] = z;
00762
00763   /* Set temperature... */
00764   if (ctl->qnt_t >= 0)
00765     atm->q[ctl->qnt_t][ip] = t;
00766
00767   /* Set zonal wind... */
00768   if (ctl->qnt_u >= 0)
00769     atm->q[ctl->qnt_u][ip] = u;
00770
00771   /* Set meridional wind... */
00772   if (ctl->qnt_v >= 0)
00773     atm->q[ctl->qnt_v][ip] = v;
00774
00775   /* Set vertical velocity... */
00776   if (ctl->qnt_w >= 0)
00777     atm->q[ctl->qnt_w][ip] = w;
00778
00779   /* Set water vapor vmr... */
00780   if (ctl->qnt_h2o >= 0)
00781     atm->q[ctl->qnt_h2o][ip] = h2o;
00782
00783   /* Set ozone vmr... */
00784   if (ctl->qnt_o3 >= 0)
00785     atm->q[ctl->qnt_o3][ip] = o3;
00786
00787   /* Calculate horizontal wind... */
00788   if (ctl->qnt_vh >= 0)
00789     atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
00790
00791   /* Calculate vertical velocity... */
00792   if (ctl->qnt_vz >= 0)
00793     atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
00794
00795   /* Calculate potential temperature... */
00796   if (ctl->qnt_theta >= 0)
00797     atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00798
00799   /* Set potential vorticity... */
00800   if (ctl->qnt_pv >= 0)
00801     atm->q[ctl->qnt_pv][ip] = pv;
00802
00803   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00804   if (ctl->qnt_tice >= 0)
00805     atm->q[ctl->qnt_tice][ip] =
00806       -2663.5 /
00807       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00808        12.537);
00809
00810   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00811   if (ctl->qnt_tnat >= 0) {
00812     if (ctl->psc_hno3 > 0)
00813       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00814     else
00815       p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
00816         * 1e-9 * atm->p[ip] / 1.333224;
00817     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00818     a = 0.009179 - 0.00088 * log10(p_h2o);
00819     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00820     c = -11397.0 / a;
00821     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00822     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00823     if (x1 > 0)
00824       atm->q[ctl->qnt_tnat][ip] = x1;
```

```
00825     if (x2 > 0)
00826       atm->q[ctl->qnt_tnat][ip] = x2;
00827   }
00828
00829   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00830   if (ctl->qnt_tsts >= 0) {
00831     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00832       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00833     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00834                                        + atm->q[ctl->qnt_tnat][ip]);
00835   }
00836 }
```

Here is the call graph for this function:



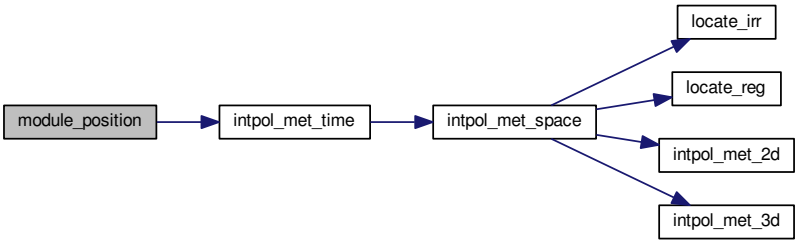
5.33.2.7 void module_position (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

Check position of air parcels.

Definition at line 840 of file trac.c.

```
00844           {
00845
00846   double ps;
00847
00848   /* Calculate modulo... */
00849   atm->lon[ip] = fmod(atm->lon[ip], 360);
00850   atm->lat[ip] = fmod(atm->lat[ip], 360);
00851
00852   /* Check latitude... */
00853   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00854     if (atm->lat[ip] > 90) {
00855       atm->lat[ip] = 180 - atm->lat[ip];
00856       atm->lon[ip] += 180;
00857     }
00858     if (atm->lat[ip] < -90) {
00859       atm->lat[ip] = -180 - atm->lat[ip];
00860       atm->lon[ip] += 180;
00861     }
00862   }
00863
00864   /* Check longitude... */
00865   while (atm->lon[ip] < -180)
00866     atm->lon[ip] += 360;
00867   while (atm->lon[ip] >= 180)
00868     atm->lon[ip] -= 360;
00869
00870   /* Get surface pressure... */
00871   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00872                   atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00873                   NULL, NULL, NULL, NULL, NULL, NULL);
00874
00875   /* Check pressure... */
00876   if (atm->p[ip] > ps)
00877     atm->p[ip] = ps;
00878   else if (atm->p[ip] < met0->p[met0->np - 1])
00879     atm->p[ip] = met0->p[met0->np - 1];
00880 }
```

Here is the call graph for this function:



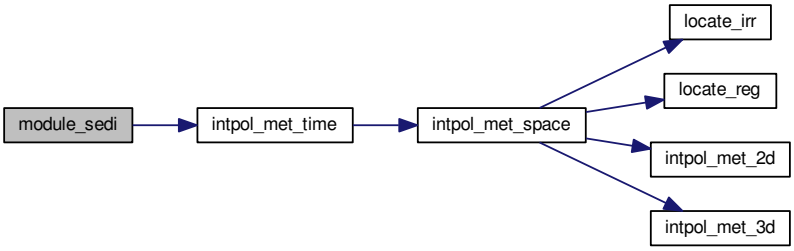
5.33.2.8 void module_sedi (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate sedimentation of air parcels.

Definition at line 884 of file trac.c.

```
00890             {
00891
00892   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00893   const double A = 1.249, B = 0.42, C = 0.87;
00894
00895   /* Average mass of an air molecule [kg/molec]: */
00896   const double m = 4.8096e-26;
00897
00898   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00899
00900   /* Convert units... */
00901   p = 100 * atm->p[ip];
00902   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00903   rho_p = atm->q[ctl->qnt_rho][ip];
00904
00905   /* Get temperature... */
00906   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00907                   atm->lat[ip], NULL, NULL, NULL, &T,
00908                   NULL, NULL, NULL, NULL, NULL, NULL);
00909
00910   /* Density of dry air... */
00911   rho = p / (RA * T);
00912
00913   /* Dynamic viscosity of air... */
00914   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00915
00916   /* Thermal velocity of an air molecule... */
00917   v = sqrt(8 * KB * T / (M_PI * m));
00918
00919   /* Mean free path of an air molecule... */
00920   lambda = 2 * eta / (rho * v);
00921
00922   /* Knudsen number for air... */
00923   K = lambda / r_p;
00924
00925   /* Cunningham slip-flow correction... */
00926   G = 1 + K * (A + B * exp(-C / K));
00927
00928   /* Sedimentation (fall) velocity... */
00929   v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
00930
00931   /* Calculate pressure change... */
00932   atm->p[ip] += DZ2DP(v_p * dt / 1000., atm->p[ip]);
00933 }
```

Here is the call graph for this function:



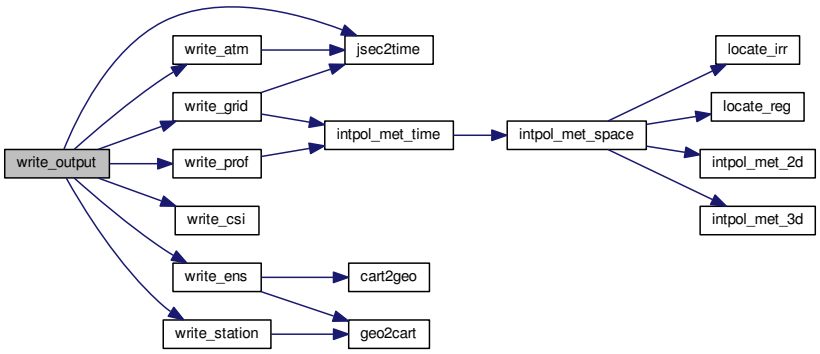
5.33.2.9 void write_output (const char * *dirname*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

Write simulation output.

Definition at line 937 of file trac.c.

```
00943             {
00944
00945   char filename[2 * LEN];
00946
00947   double r;
00948
00949   int year, mon, day, hour, min, sec;
00950
00951   /* Get time... */
00952   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00953
00954   /* Write atmospheric data... */
00955   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
00956     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00957             dirname, ctl->atm_basename, year, mon, day, hour, min);
00958     write_atm(filename, ctl, atm, t);
00959   }
00960
00961   /* Write CSI data... */
00962   if (ctl->csi_basename[0] != '-') {
00963     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00964     write_csi(filename, ctl, atm, t);
00965   }
00966
00967   /* Write ensemble data... */
00968   if (ctl->ens_basename[0] != '-') {
00969     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
00970     write_ens(filename, ctl, atm, t);
00971   }
00972
00973   /* Write gridded data... */
00974   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
00975     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00976             dirname, ctl->grid_basename, year, mon, day, hour, min);
00977     write_grid(filename, ctl, met0, met1, atm, t);
00978   }
00979
00980   /* Write profile data... */
00981   if (ctl->prof_basename[0] != '-') {
00982     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00983     write_prof(filename, ctl, met0, met1, atm, t);
00984   }
00985
00986   /* Write station data... */
00987   if (ctl->stat_basename[0] != '-') {
00988     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00989     write_station(filename, ctl, atm, t);
00990   }
00991 }
```

Here is the call graph for this function:



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

Definition at line 115 of file trac.c.

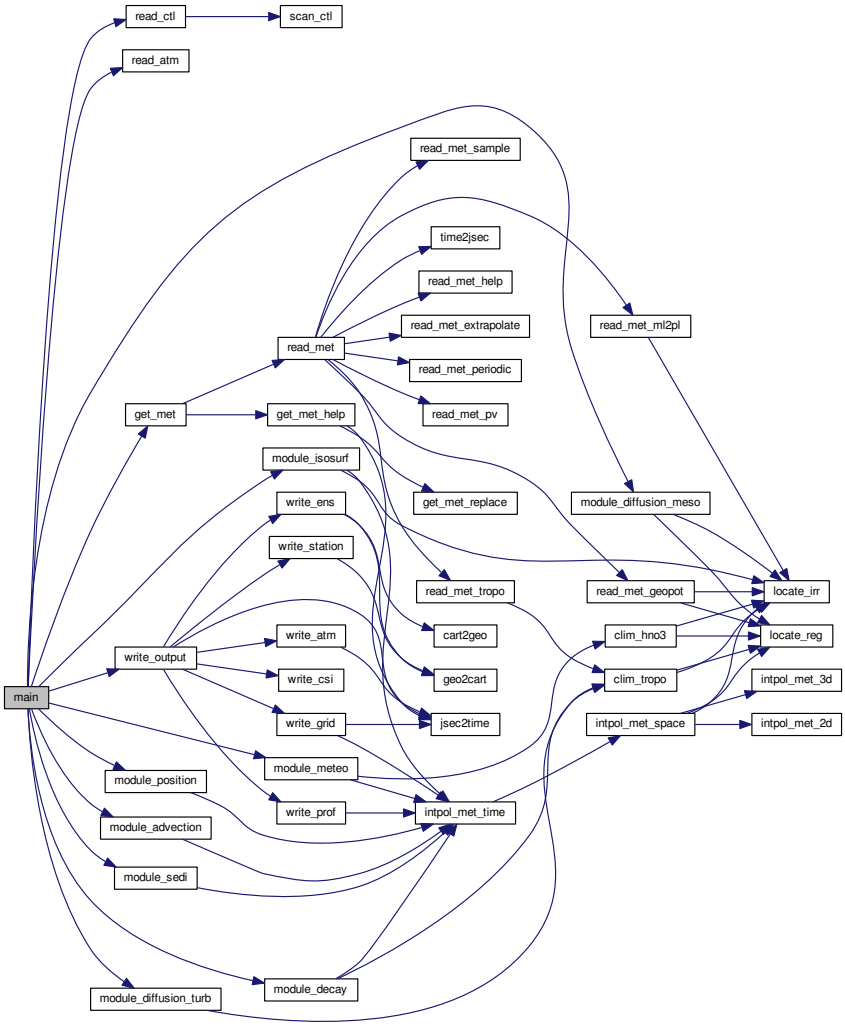
```
00117                {
00118
00119   ctl_t ctl;
00120
00121   atm_t *atm;
00122
00123   met_t *met0, *met1;
00124
00125   gsl_rng *rng[NTHREADS];
00126
00127   FILE *dirlist;
00128
00129   char dirname[LEN], filename[2 * LEN];
00130
00131   double *dt, t;
00132
00133   int i, ip, ntask = -1, rank = 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     if (!read_atm(filename, &ctl, atm))
00188       ERRMSG("Cannot open file!");
00189
00190     /* Set start time... */
00191     if (ctl.direction == 1) {
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 {
00196       ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00197       if (ctl.t_stop > 1e99)
00198         ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00199     }
00200
00201     /* Check time interval... */
00202     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00203       ERRMSG("Nothing to do!");
00204
00205     /* Round start time... */
00206     if (ctl.direction == 1)
00207       ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
      dt_mod;
00208     else
00209       ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
      dt_mod;
00210
00211     /* Set timers... */
00212     STOP_TIMER(TIMER_INIT);
00213
00214     /* ------------------------------------------------------------
00215        Loop over timesteps...
00216        ------------------------------------------------------------ */
00217
00218     /* Loop over timesteps... */
00219     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
      dt_mod;
00220          t += ctl.direction * ctl.dt_mod) {
00221
00222       /* Adjust length of final time step... */
00223       if (ctl.direction * (t - ctl.t_stop) > 0)
00224         t = ctl.t_stop;
00225
00226       /* Set time steps for air parcels... */
00227       for (ip = 0; ip < atm->np; ip++)
00228         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00229              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00230              && ctl.direction * (atm->time[ip] - t) < 0))
00231           dt[ip] = t - atm->time[ip];
00232         else
00233           dt[ip] = GSL_NAN;
00234
00235       /* Get meteorological data... */
00236       START_TIMER(TIMER_INPUT);
00237       get_met(&ctl, argv[4], t, &met0, &met1);
00238       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00239         printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00240                fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00241       STOP_TIMER(TIMER_INPUT);
00242
00243       /* Initialize isosurface... */
00244       START_TIMER(TIMER_ISOSURF);
00245       if (ctl.isosurf >= 1 && ctl.isosurf <= 4 && t == ctl.t_start)
00246         module_isosurf(&ctl, met0, met1, atm, -1);
00247       STOP_TIMER(TIMER_ISOSURF);
00248
00249       /* Advection... */
00250       START_TIMER(TIMER_ADVECT);
```

```
00251 #pragma omp parallel for default(shared) private(ip)
00252       for (ip = 0; ip < atm->np; ip++)
00253         if (gsl_finite(dt[ip]))
00254           module_advection(met0, met1, atm, ip, dt[ip]);
00255       STOP_TIMER(TIMER_ADVECT);
00256
00257       /* Turbulent diffusion... */
00258       START_TIMER(TIMER_DIFFTURB);
00259       if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00260           || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00261 #pragma omp parallel for default(shared) private(ip)
00262         for (ip = 0; ip < atm->np; ip++)
00263           if (gsl_finite(dt[ip]))
00264             module_diffusion_turb(&ctl, atm, ip, dt[ip],
00265                                   rng[omp_get_thread_num()]);
00266       }
00267       STOP_TIMER(TIMER_DIFFTURB);
00268
00269       /* Mesoscale diffusion... */
00270       START_TIMER(TIMER_DIFFMESO);
00271       if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00272 #pragma omp parallel for default(shared) private(ip)
00273         for (ip = 0; ip < atm->np; ip++)
00274           if (gsl_finite(dt[ip]))
00275             module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00276                                   rng[omp_get_thread_num()]);
00277       }
00278       STOP_TIMER(TIMER_DIFFMESO);
00279
00280       /* Sedimentation... */
00281       START_TIMER(TIMER_SEDI);
00282       if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
00283 #pragma omp parallel for default(shared) private(ip)
00284         for (ip = 0; ip < atm->np; ip++)
00285           if (gsl_finite(dt[ip]))
00286             module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00287       }
00288       STOP_TIMER(TIMER_SEDI);
00289
00290       /* Isosurface... */
00291       START_TIMER(TIMER_ISOSURF);
00292       if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
00293 #pragma omp parallel for default(shared) private(ip)
00294         for (ip = 0; ip < atm->np; ip++)
00295           module_isosurf(&ctl, met0, met1, atm, ip);
00296       }
00297       STOP_TIMER(TIMER_ISOSURF);
00298
00299       /* Position... */
00300       START_TIMER(TIMER_POSITION);
00301 #pragma omp parallel for default(shared) private(ip)
00302       for (ip = 0; ip < atm->np; ip++)
00303         module_position(met0, met1, atm, ip);
00304       STOP_TIMER(TIMER_POSITION);
00305
00306       /* Meteorological data... */
00307       START_TIMER(TIMER_METEO);
00308       if (ctl.met_dt_out > 0
00309           && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
      met_dt_out) == 0)) {
00310 #pragma omp parallel for default(shared) private(ip)
00311         for (ip = 0; ip < atm->np; ip++)
00312           module_meteo(&ctl, met0, met1, atm, ip);
00313       }
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_INPUT);
00356     PRINT_TIMER(TIMER_OUTPUT);
00357     PRINT_TIMER(TIMER_ADVECT);
00358     PRINT_TIMER(TIMER_DECAY);
00359     PRINT_TIMER(TIMER_DIFFMESO);
00360     PRINT_TIMER(TIMER_DIFFTURB);
00361     PRINT_TIMER(TIMER_ISOSURF);
00362     PRINT_TIMER(TIMER_METEO);
00363     PRINT_TIMER(TIMER_POSITION);
00364     PRINT_TIMER(TIMER_SEDI);
00365
00366     /* Free random number generators... */
00367     for (i = 0; i < NTHREADS; i++)
00368       gsl_rng_free(rng[i]);
00369
00370     /* Free... */
00371     free(atm);
00372     free(met0);
00373     free(met1);
00374     free(dt);
00375   }
00376
00377 #ifdef MPI
00378   /* Finalize MPI... */
00379   MPI_Finalize();
00380 #endif
00381
00382   return EXIT_SUCCESS;
00383 }
```

Here is the call graph for this function:



5.34 trac.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 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[2 * LEN];
00130
00131   double *dt, t;
00132
00133   int i, ip, ntask = -1, 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     if (!read_atm(filename, &ctl, atm))
00188       ERRMSG("Cannot open file!");
00189
00190     /* Set start time... */
00191     if (ctl.direction == 1) {
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 {
00196       ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00197       if (ctl.t_stop > 1e99)
00198         ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00199     }
00200
00201     /* Check time interval... */
00202     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00203       ERRMSG("Nothing to do!");
00204
00205     /* Round start time... */
00206     if (ctl.direction == 1)
00207       ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
      dt_mod;
00208     else
00209       ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
      dt_mod;
```

```
00210
00211     /* Set timers... */
00212     STOP_TIMER(TIMER_INIT);
00213
00214     /* ------------------------------------------------------------
00215        Loop over timesteps...
00216        ------------------------------------------------------------ */
00217
00218     /* Loop over timesteps... */
00219     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
      dt_mod;
00220          t += ctl.direction * ctl.dt_mod) {
00221
00222       /* Adjust length of final time step... */
00223       if (ctl.direction * (t - ctl.t_stop) > 0)
00224         t = ctl.t_stop;
00225
00226       /* Set time steps for air parcels... */
00227       for (ip = 0; ip < atm->np; ip++)
00228         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00229              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00230              && ctl.direction * (atm->time[ip] - t) < 0))
00231           dt[ip] = t - atm->time[ip];
00232         else
00233           dt[ip] = GSL_NAN;
00234
00235       /* Get meteorological data... */
00236       START_TIMER(TIMER_INPUT);
00237       get_met(&ctl, argv[4], t, &met0, &met1);
00238       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00239         printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00240                fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00241       STOP_TIMER(TIMER_INPUT);
00242
00243       /* Initialize isosurface... */
00244       START_TIMER(TIMER_ISOSURF);
00245       if (ctl.isosurf >= 1 && ctl.isosurf <= 4 && t == ctl.t_start)
00246         module_isosurf(&ctl, met0, met1, atm, -1);
00247       STOP_TIMER(TIMER_ISOSURF);
00248
00249       /* Advection... */
00250       START_TIMER(TIMER_ADVECT);
00251 #pragma omp parallel for default(shared) private(ip)
00252       for (ip = 0; ip < atm->np; ip++)
00253         if (gsl_finite(dt[ip]))
00254           module_advection(met0, met1, atm, ip, dt[ip]);
00255       STOP_TIMER(TIMER_ADVECT);
00256
00257       /* Turbulent diffusion... */
00258       START_TIMER(TIMER_DIFFTURB);
00259       if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00260           || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00261 #pragma omp parallel for default(shared) private(ip)
00262         for (ip = 0; ip < atm->np; ip++)
00263           if (gsl_finite(dt[ip]))
00264             module_diffusion_turb(&ctl, atm, ip, dt[ip],
00265                                   rng[omp_get_thread_num()]);
00266       }
00267       STOP_TIMER(TIMER_DIFFTURB);
00268
00269       /* Mesoscale diffusion... */
00270       START_TIMER(TIMER_DIFFMESO);
00271       if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00272 #pragma omp parallel for default(shared) private(ip)
00273         for (ip = 0; ip < atm->np; ip++)
00274           if (gsl_finite(dt[ip]))
00275             module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00276                                   rng[omp_get_thread_num()]);
00277       }
00278       STOP_TIMER(TIMER_DIFFMESO);
00279
00280       /* Sedimentation... */
00281       START_TIMER(TIMER_SEDI);
00282       if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
00283 #pragma omp parallel for default(shared) private(ip)
00284         for (ip = 0; ip < atm->np; ip++)
00285           if (gsl_finite(dt[ip]))
00286             module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00287       }
00288       STOP_TIMER(TIMER_SEDI);
00289
00290       /* Isosurface... */
00291       START_TIMER(TIMER_ISOSURF);
00292       if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
00293 #pragma omp parallel for default(shared) private(ip)
00294         for (ip = 0; ip < atm->np; ip++)
00295           module_isosurf(&ctl, met0, met1, atm, ip);
```

```
00296       }
00297       STOP_TIMER(TIMER_ISOSURF);
00298
00299       /* Position... */
00300       START_TIMER(TIMER_POSITION);
00301 #pragma omp parallel for default(shared) private(ip)
00302       for (ip = 0; ip < atm->np; ip++)
00303         module_position(met0, met1, atm, ip);
00304       STOP_TIMER(TIMER_POSITION);
00305
00306       /* Meteorological data... */
00307       START_TIMER(TIMER_METEO);
00308       if (ctl.met_dt_out > 0
00309           && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
      met_dt_out) == 0)) {
00310 #pragma omp parallel for default(shared) private(ip)
00311         for (ip = 0; ip < atm->np; ip++)
00312           module_meteo(&ctl, met0, met1, atm, ip);
00313       }
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_INPUT);
00356     PRINT_TIMER(TIMER_OUTPUT);
00357     PRINT_TIMER(TIMER_ADVECT);
00358     PRINT_TIMER(TIMER_DECAY);
00359     PRINT_TIMER(TIMER_DIFFMESO);
00360     PRINT_TIMER(TIMER_DIFFTURB);
00361     PRINT_TIMER(TIMER_ISOSURF);
00362     PRINT_TIMER(TIMER_METEO);
00363     PRINT_TIMER(TIMER_POSITION);
00364     PRINT_TIMER(TIMER_SEDI);
00365
00366     /* Free random number generators... */
00367     for (i = 0; i < NTHREADS; i++)
00368       gsl_rng_free(rng[i]);
00369
00370     /* Free... */
00371     free(atm);
00372     free(met0);
00373     free(met1);
00374     free(dt);
00375   }
00376
00377 #ifdef MPI
00378   /* Finalize MPI... */
00379   MPI_Finalize();
00380 #endif
```

```
00381
00382   return EXIT_SUCCESS;
00383 }
00384
00385 /*****************************************************************************/
00386
00387 void module_advection(
00388   met_t * met0,
00389   met_t * met1,
00390   atm_t * atm,
00391   int ip,
00392   double dt) {
00393
00394   double v[3], xm[3];
00395
00396   /* Interpolate meteorological data... */
00397   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00398                   atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
00399                   &v[0], &v[1], &v[2], NULL, NULL, NULL);
00400
00401   /* Get position of the mid point... */
00402   xm[0] = atm->lon[ip] + DX2DEG(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00403   xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt * v[1] / 1000.);
00404   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00405
00406   /* Interpolate meteorological data for mid point... */
00407   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00408                   xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
00409                   &v[0], &v[1], &v[2], NULL, NULL, NULL);
00410
00411   /* Save new position... */
00412   atm->time[ip] += dt;
00413   atm->lon[ip] += DX2DEG(dt * v[0] / 1000., xm[1]);
00414   atm->lat[ip] += DY2DEG(dt * v[1] / 1000.);
00415   atm->p[ip] += dt * v[2];
00416 }
00417
00418 /*****************************************************************************/
00419
00420 void module_decay(
00421   ctl_t * ctl,
00422   met_t * met0,
00423   met_t * met1,
00424   atm_t * atm,
00425   int ip,
00426   double dt) {
00427
00428   double ps, pt, tdec;
00429
00430   /* Set constant lifetime... */
00431   if (ctl->tdec_trop == ctl->tdec_strat)
00432     tdec = ctl->tdec_trop;
00433
00434   /* Set altitude-dependent lifetime... */
00435   else {
00436
00437     /* Get surface pressure... */
00438     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00439                     atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00440                     NULL, NULL, NULL, NULL, NULL, NULL);
00441
00442     /* Get tropopause pressure... */
00443     pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00444
00445     /* Set lifetime... */
00446     if (atm->p[ip] <= pt)
00447       tdec = ctl->tdec_strat;
00448     else
00449       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00450   }
00451
00452   /* Calculate exponential decay... */
00453   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00454 }
00455
00456 /*****************************************************************************/
00457
00458 void module_diffusion_meso(
00459   ctl_t * ctl,
00460   met_t * met0,
00461   met_t * met1,
00462   atm_t * atm,
00463   int ip,
00464   double dt,
00465   gsl_rng * rng) {
00466
```

```
00467   double r, rs, u[16], v[16], w[16];
00468
00469   int ix, iy, iz;
00470
00471   /* Get indices... */
00472   ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00473   iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00474   iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00475
00476   /* Caching of wind standard deviations... */
00477   if (atm->cache_time[ix][iy][iz] != met0->time) {
00478
00479     /* Collect local wind data... */
00480     u[0] = met0->u[ix][iy][iz];
00481     u[1] = met0->u[ix + 1][iy][iz];
00482     u[2] = met0->u[ix][iy + 1][iz];
00483     u[3] = met0->u[ix + 1][iy + 1][iz];
00484     u[4] = met0->u[ix][iy][iz + 1];
00485     u[5] = met0->u[ix + 1][iy][iz + 1];
00486     u[6] = met0->u[ix][iy + 1][iz + 1];
00487     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00488
00489     v[0] = met0->v[ix][iy][iz];
00490     v[1] = met0->v[ix + 1][iy][iz];
00491     v[2] = met0->v[ix][iy + 1][iz];
00492     v[3] = met0->v[ix + 1][iy + 1][iz];
00493     v[4] = met0->v[ix][iy][iz + 1];
00494     v[5] = met0->v[ix + 1][iy][iz + 1];
00495     v[6] = met0->v[ix][iy + 1][iz + 1];
00496     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00497
00498     w[0] = met0->w[ix][iy][iz];
00499     w[1] = met0->w[ix + 1][iy][iz];
00500     w[2] = met0->w[ix][iy + 1][iz];
00501     w[3] = met0->w[ix + 1][iy + 1][iz];
00502     w[4] = met0->w[ix][iy][iz + 1];
00503     w[5] = met0->w[ix + 1][iy][iz + 1];
00504     w[6] = met0->w[ix][iy + 1][iz + 1];
00505     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00506
00507     /* Collect local wind data... */
00508     u[8] = met1->u[ix][iy][iz];
00509     u[9] = met1->u[ix + 1][iy][iz];
00510     u[10] = met1->u[ix][iy + 1][iz];
00511     u[11] = met1->u[ix + 1][iy + 1][iz];
00512     u[12] = met1->u[ix][iy][iz + 1];
00513     u[13] = met1->u[ix + 1][iy][iz + 1];
00514     u[14] = met1->u[ix][iy + 1][iz + 1];
00515     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00516
00517     v[8] = met1->v[ix][iy][iz];
00518     v[9] = met1->v[ix + 1][iy][iz];
00519     v[10] = met1->v[ix][iy + 1][iz];
00520     v[11] = met1->v[ix + 1][iy + 1][iz];
00521     v[12] = met1->v[ix][iy][iz + 1];
00522     v[13] = met1->v[ix + 1][iy][iz + 1];
00523     v[14] = met1->v[ix][iy + 1][iz + 1];
00524     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00525
00526     w[8] = met1->w[ix][iy][iz];
00527     w[9] = met1->w[ix + 1][iy][iz];
00528     w[10] = met1->w[ix][iy + 1][iz];
00529     w[11] = met1->w[ix + 1][iy + 1][iz];
00530     w[12] = met1->w[ix][iy][iz + 1];
00531     w[13] = met1->w[ix + 1][iy][iz + 1];
00532     w[14] = met1->w[ix][iy + 1][iz + 1];
00533     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00534
00535     /* Get standard deviations of local wind data... */
00536     atm->cache_usig[ix][iy][iz] = (float) gsl_stats_sd(u, 1, 16);
00537     atm->cache_vsig[ix][iy][iz] = (float) gsl_stats_sd(v, 1, 16);
00538     atm->cache_wsig[ix][iy][iz] = (float) gsl_stats_sd(w, 1, 16);
00539     atm->cache_time[ix][iy][iz] = met0->time;
00540   }
00541
00542   /* Set temporal correlations for mesoscale fluctuations... */
00543   r = 1 - 2 * fabs(dt) / ctl->dt_met;
00544   rs = sqrt(1 - r * r);
00545
00546   /* Calculate horizontal mesoscale wind fluctuations... */
00547   if (ctl->turb_mesox > 0) {
00548     atm->up[ip] = (float)
00549       (r * atm->up[ip]
00550        + rs * gsl_ran_gaussian_ziggurat(rng,
00551                                         ctl->turb_mesox *
00552                                         atm->cache_usig[ix][iy][iz]));
00553     atm->lon[ip] += DX2DEG(atm->up[ip] * dt / 1000., atm->lat[ip]);
```

```
00554
00555     atm->vp[ip] = (float)
00556       (r * atm->vp[ip]
00557        + rs * gsl_ran_gaussian_ziggurat(rng,
00558                                         ctl->turb_mesox *
00559                                         atm->cache_vsig[ix][iy][iz]));
00560     atm->lat[ip] += DY2DEG(atm->vp[ip] * dt / 1000.);
00561   }
00562
00563   /* Calculate vertical mesoscale wind fluctuations... */
00564   if (ctl->turb_mesoz > 0) {
00565     atm->wp[ip] = (float)
00566       (r * atm->wp[ip]
00567        + rs * gsl_ran_gaussian_ziggurat(rng,
00568                                         ctl->turb_mesoz *
00569                                         atm->cache_wsig[ix][iy][iz]));
00570     atm->p[ip] += atm->wp[ip] * dt;
00571   }
00572 }
00573
00574 /*****************************************************************************/
00575
00576 void module_diffusion_turb(
00577   ctl_t * ctl,
00578   atm_t * atm,
00579   int ip,
00580   double dt,
00581   gsl_rng * rng) {
00582
00583   double dx, dz, pt, p0, p1, w;
00584
00585   /* Get tropopause pressure... */
00586   pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00587
00588   /* Get weighting factor... */
00589   p1 = pt * 0.866877899;
00590   p0 = pt / 0.866877899;
00591   if (atm->p[ip] > p0)
00592     w = 1;
00593   else if (atm->p[ip] < p1)
00594     w = 0;
00595   else
00596     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00597
00598   /* Set diffusivity... */
00599   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00600   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00601
00602   /* Horizontal turbulent diffusion... */
00603   if (dx > 0) {
00604     atm->lon[ip]
00605       += DX2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00606                 / 1000., atm->lat[ip]);
00607     atm->lat[ip]
00608       += DY2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00609                 / 1000.);
00610   }
00611
00612   /* Vertical turbulent diffusion... */
00613   if (dz > 0)
00614     atm->p[ip]
00615       += DZ2DP(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00616                / 1000., atm->p[ip]);
00617 }
00618
00619 /*****************************************************************************/
00620
00621 void module_isosurf(
00622   ctl_t * ctl,
00623   met_t * met0,
00624   met_t * met1,
00625   atm_t * atm,
00626   int ip) {
00627
00628   static double *iso, *ps, t, *ts;
00629
00630   static int idx, ip2, n;
00631
00632   FILE *in;
00633
00634   char line[LEN];
00635
00636   /* Initialize... */
00637   if (ip < 0) {
00638
00639     /* Allocate... */
00640     ALLOC(iso, double,
```

```
00641           NP);
00642     ALLOC(ps, double,
00643           NP);
00644     ALLOC(ts, double,
00645           NP);
00646
00647     /* Save pressure... */
00648     if (ctl->isosurf == 1)
00649       for (ip2 = 0; ip2 < atm->np; ip2++)
00650         iso[ip2] = atm->p[ip2];
00651
00652     /* Save density... */
00653     else if (ctl->isosurf == 2)
00654       for (ip2 = 0; ip2 < atm->np; ip2++) {
00655         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00656                         atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00657                         &t, NULL, NULL, NULL, NULL, NULL, NULL);
00658         iso[ip2] = atm->p[ip2] / t;
00659       }
00660
00661     /* Save potential temperature... */
00662     else if (ctl->isosurf == 3)
00663       for (ip2 = 0; ip2 < atm->np; ip2++) {
00664         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00665                         atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
00666                         &t, NULL, NULL, NULL, NULL, NULL, NULL);
00667         iso[ip2] = THETA(atm->p[ip2], t);
00668       }
00669
00670     /* Read balloon pressure data... */
00671     else if (ctl->isosurf == 4) {
00672
00673       /* Write info... */
00674       printf("Read balloon pressure data: %s\n", ctl->balloon);
00675
00676       /* Open file... */
00677       if (!(in = fopen(ctl->balloon, "r")))
00678         ERRMSG("Cannot open file!");
00679
00680       /* Read pressure time series... */
00681       while (fgets(line, LEN, in))
00682         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00683           if ((++n) > NP)
00684             ERRMSG("Too many data points!");
00685
00686       /* Check number of points... */
00687       if (n < 1)
00688         ERRMSG("Could not read any data!");
00689
00690       /* Close file... */
00691       fclose(in);
00692     }
00693
00694     /* Leave initialization... */
00695     return;
00696   }
00697
00698   /* Restore pressure... */
00699   if (ctl->isosurf == 1)
00700     atm->p[ip] = iso[ip];
00701
00702   /* Restore density... */
00703   else if (ctl->isosurf == 2) {
00704     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00705                     atm->lat[ip], NULL, NULL, NULL, &t,
00706                     NULL, NULL, NULL, NULL, NULL, NULL);
00707     atm->p[ip] = iso[ip] * t;
00708   }
00709
00710   /* Restore potential temperature... */
00711   else if (ctl->isosurf == 3) {
00712     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00713                     atm->lat[ip], NULL, NULL, NULL, &t,
00714                     NULL, NULL, NULL, NULL, NULL, NULL);
00715     atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
00716   }
00717
00718   /* Interpolate pressure... */
00719   else if (ctl->isosurf == 4) {
00720     if (atm->time[ip] <= ts[0])
00721       atm->p[ip] = ps[0];
00722     else if (atm->time[ip] >= ts[n - 1])
00723       atm->p[ip] = ps[n - 1];
00724     else {
00725       idx = locate_irr(ts, n, atm->time[ip]);
```

```
00726       atm->p[ip] = LIN(ts[idx], ps[idx],
00727                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00728     }
00729   }
00730 }
00731
00732 /*****************************************************************************/
00733
00734 void module_meteo(
00735   ctl_t * ctl,
00736   met_t * met0,
00737   met_t * met1,
00738   atm_t * atm,
00739   int ip) {
00740
00741   double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;
00742
00743   /* Interpolate meteorological data... */
00744   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00745                   atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);
00746
00747   /* Set surface pressure... */
00748   if (ctl->qnt_ps >= 0)
00749     atm->q[ctl->qnt_ps][ip] = ps;
00750
00751   /* Set tropopause pressure... */
00752   if (ctl->qnt_pt >= 0)
00753     atm->q[ctl->qnt_pt][ip] = pt;
00754
00755   /* Set pressure... */
00756   if (ctl->qnt_p >= 0)
00757     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00758
00759   /* Set geopotential height... */
00760   if (ctl->qnt_z >= 0)
00761     atm->q[ctl->qnt_z][ip] = z;
00762
00763   /* Set temperature... */
00764   if (ctl->qnt_t >= 0)
00765     atm->q[ctl->qnt_t][ip] = t;
00766
00767   /* Set zonal wind... */
00768   if (ctl->qnt_u >= 0)
00769     atm->q[ctl->qnt_u][ip] = u;
00770
00771   /* Set meridional wind... */
00772   if (ctl->qnt_v >= 0)
00773     atm->q[ctl->qnt_v][ip] = v;
00774
00775   /* Set vertical velocity... */
00776   if (ctl->qnt_w >= 0)
00777     atm->q[ctl->qnt_w][ip] = w;
00778
00779   /* Set water vapor vmr... */
00780   if (ctl->qnt_h2o >= 0)
00781     atm->q[ctl->qnt_h2o][ip] = h2o;
00782
00783   /* Set ozone vmr... */
00784   if (ctl->qnt_o3 >= 0)
00785     atm->q[ctl->qnt_o3][ip] = o3;
00786
00787   /* Calculate horizontal wind... */
00788   if (ctl->qnt_vh >= 0)
00789     atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
00790
00791   /* Calculate vertical velocity... */
00792   if (ctl->qnt_vz >= 0)
00793     atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
00794
00795   /* Calculate potential temperature... */
00796   if (ctl->qnt_theta >= 0)
00797     atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
00798
00799   /* Set potential vorticity... */
00800   if (ctl->qnt_pv >= 0)
00801     atm->q[ctl->qnt_pv][ip] = pv;
00802
00803   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00804   if (ctl->qnt_tice >= 0)
00805     atm->q[ctl->qnt_tice][ip] =
00806       -2663.5 /
00807       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00808        12.537);
00809
00810   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00811   if (ctl->qnt_tnat >= 0) {
```

```
00812     if (ctl->psc_hno3 > 0)
00813       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00814     else
00815       p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
00816         * 1e-9 * atm->p[ip] / 1.333224;
00817     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00818     a = 0.009179 - 0.00088 * log10(p_h2o);
00819     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00820     c = -11397.0 / a;
00821     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00822     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00823     if (x1 > 0)
00824       atm->q[ctl->qnt_tnat][ip] = x1;
00825     if (x2 > 0)
00826       atm->q[ctl->qnt_tnat][ip] = x2;
00827   }
00828
00829   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00830   if (ctl->qnt_tsts >= 0) {
00831     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00832       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00833     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00834                                        + atm->q[ctl->qnt_tnat][ip]);
00835   }
00836 }
00837
00838 /*****************************************************************************/
00839
00840 void module_position(
00841   met_t * met0,
00842   met_t * met1,
00843   atm_t * atm,
00844   int ip) {
00845
00846   double ps;
00847
00848   /* Calculate modulo... */
00849   atm->lon[ip] = fmod(atm->lon[ip], 360);
00850   atm->lat[ip] = fmod(atm->lat[ip], 360);
00851
00852   /* Check latitude... */
00853   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00854     if (atm->lat[ip] > 90) {
00855       atm->lat[ip] = 180 - atm->lat[ip];
00856       atm->lon[ip] += 180;
00857     }
00858     if (atm->lat[ip] < -90) {
00859       atm->lat[ip] = -180 - atm->lat[ip];
00860       atm->lon[ip] += 180;
00861     }
00862   }
00863
00864   /* Check longitude... */
00865   while (atm->lon[ip] < -180)
00866     atm->lon[ip] += 360;
00867   while (atm->lon[ip] >= 180)
00868     atm->lon[ip] -= 360;
00869
00870   /* Get surface pressure... */
00871   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00872                   atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
00873                   NULL, NULL, NULL, NULL, NULL, NULL);
00874
00875   /* Check pressure... */
00876   if (atm->p[ip] > ps)
00877     atm->p[ip] = ps;
00878   else if (atm->p[ip] < met0->p[met0->np - 1])
00879     atm->p[ip] = met0->p[met0->np - 1];
00880 }
00881
00882 /*****************************************************************************/
00883
00884 void module_sedi(
00885   ctl_t * ctl,
00886   met_t * met0,
00887   met_t * met1,
00888   atm_t * atm,
00889   int ip,
00890   double dt) {
00891
00892   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00893   const double A = 1.249, B = 0.42, C = 0.87;
00894
00895   /* Average mass of an air molecule [kg/molec]: */
00896   const double m = 4.8096e-26;
00897
00898   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
```

```
00899
00900   /* Convert units... */
00901   p = 100 * atm->p[ip];
00902   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00903   rho_p = atm->q[ctl->qnt_rho][ip];
00904
00905   /* Get temperature... */
00906   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00907                   atm->lat[ip], NULL, NULL, NULL, &T,
00908                   NULL, NULL, NULL, NULL, NULL, NULL);
00909
00910   /* Density of dry air... */
00911   rho = p / (RA * T);
00912
00913   /* Dynamic viscosity of air... */
00914   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00915
00916   /* Thermal velocity of an air molecule... */
00917   v = sqrt(8 * KB * T / (M_PI * m));
00918
00919   /* Mean free path of an air molecule... */
00920   lambda = 2 * eta / (rho * v);
00921
00922   /* Knudsen number for air... */
00923   K = lambda / r_p;
00924
00925   /* Cunningham slip-flow correction... */
00926   G = 1 + K * (A + B * exp(-C / K));
00927
00928   /* Sedimentation (fall) velocity... */
00929   v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
00930
00931   /* Calculate pressure change... */
00932   atm->p[ip] += DZ2DP(v_p * dt / 1000., atm->p[ip]);
00933 }
00934
00935 /*****************************************************************************/
00936
00937 void write_output(
00938   const char *dirname,
00939   ctl_t * ctl,
00940   met_t * met0,
00941   met_t * met1,
00942   atm_t * atm,
00943   double t) {
00944
00945   char filename[2 * LEN];
00946
00947   double r;
00948
00949   int year, mon, day, hour, min, sec;
00950
00951   /* Get time... */
00952   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
00953
00954   /* Write atmospheric data... */
00955   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
00956     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00957             dirname, ctl->atm_basename, year, mon, day, hour, min);
00958     write_atm(filename, ctl, atm, t);
00959   }
00960
00961   /* Write CSI data... */
00962   if (ctl->csi_basename[0] != '-') {
00963     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
00964     write_csi(filename, ctl, atm, t);
00965   }
00966
00967   /* Write ensemble data... */
00968   if (ctl->ens_basename[0] != '-') {
00969     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
00970     write_ens(filename, ctl, atm, t);
00971   }
00972
00973   /* Write gridded data... */
00974   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
00975     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
00976             dirname, ctl->grid_basename, year, mon, day, hour, min);
00977     write_grid(filename, ctl, met0, met1, atm, t);
00978   }
00979
00980   /* Write profile data... */
00981   if (ctl->prof_basename[0] != '-') {
00982     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
00983     write_prof(filename, ctl, met0, met1, atm, t);
00984   }
```

```
00985
00986   /* Write station data... */
00987   if (ctl->stat_basename[0] != '-') {
00988     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
00989     write_station(filename, ctl, atm, t);
00990   }
00991 }
```

Index