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.

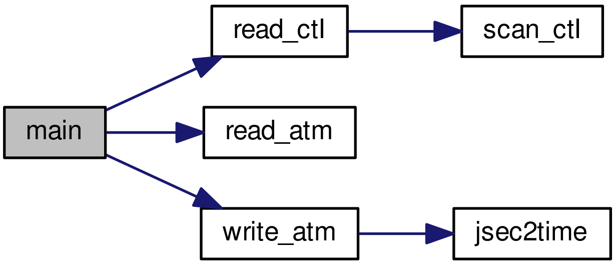
5.1.2 Function Documentation

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

Definition at line 27 of file atm_conv.c.

```
                {

  ctl_t ctl;

  atm_t *atm;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
           " <atm_out> <atm_out_type>");

  /* Allocate... */
  ALLOC(atm, atm_t, 1);

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);

  /* Read atmospheric data... */
  ctl.atm_type = atoi(argv[3]);
  if (!read_atm(argv[2], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Write atmospheric data... */
  ctl.atm_type = atoi(argv[5]);
  write_atm(argv[4], &ctl, atm, 0);

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.2 atm_conv.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"
```

```

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

  ctl_t ctl;

  atm_t *atm;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
           " <atm_out> <atm_out_type>");

  /* Allocate... */
  ALLOC(atm, atm_t, 1);

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);

  /* Read atmospheric data... */
  ctl.atm_type = atoi(argv[3]);
  if (!read_atm(argv[2], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Write atmospheric data... */
  ctl.atm_type = atoi(argv[5]);
  write_atm(argv[4], &ctl, atm, 0);

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

  return EXIT_SUCCESS;
}
```

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[ ]*)

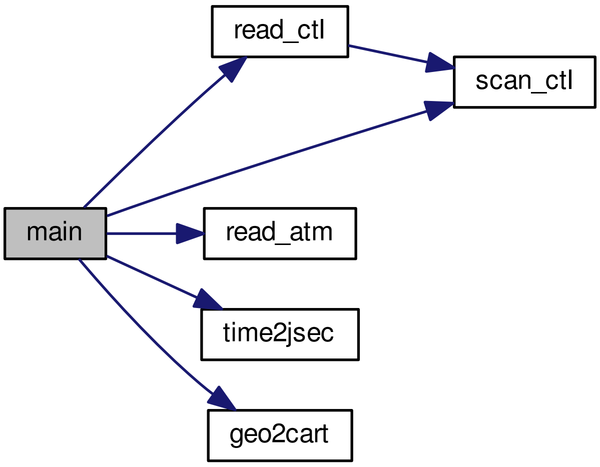
Definition at line 27 of file atm_dist.c.

```
                {

  ctl_t ctl;

  atm_t *atm1, *atm2;

  FILE *out;

  char tstr[LEN];

```

```
double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
  *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
  *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
  t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;

int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;

/* Allocate... */
ALLOC(atm1, atm_t, 1);
ALLOC(atm2, atm_t, 1);
ALLOC(lon1_old, double,
      NP);
ALLOC(lat1_old, double,
      NP);
ALLOC(z1_old, double,
      NP);
ALLOC(lh1, double,
      NP);
ALLOC(lv1, double,
      NP);
ALLOC(lon2_old, double,
      NP);
ALLOC(lat2_old, double,
      NP);
ALLOC(z2_old, double,
      NP);
ALLOC(lh2, double,
      NP);
ALLOC(lv2, double,
      NP);
ALLOC(ahtd, double,
      NP);
ALLOC(avtd, double,
      NP);
ALLOC(aqtd, double,
      NP * NQ);
ALLOC(rhtd, double,
      NP);
ALLOC(rvtd, double,
      NP);
ALLOC(rqtd, double,
      NP * NQ);
ALLOC(work, double,
      NP);

/* Check arguments... */
if (argc < 6)
  ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atm1a> <atm1b>"
         " [<atm2a> <atm2b> ...]");

/* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);

/* Write info... */
printf("Write transport deviations: %s\n", argv[2]);

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

/* Write header... */
fprintf(out,
        "# $1 = time [s]\n"
        "# $2 = time difference [s]\n"
        "# $3 = absolute horizontal distance (%s) [km]\n"
        "# $4 = relative horizontal distance (%s) [%%]\n"
        "# $5 = absolute vertical distance (%s) [km]\n"
        "# $6 = relative vertical distance (%s) [%%]\n",
        argv[3], argv[3], argv[3], argv[3]);
for (iq = 0; iq < ctl.nq; iq++)
  fprintf(out,
          "# $%d = %s absolute difference (%s) [%s]\n"
          "# $%d = %s relative difference (%s) [%%]\n",
          7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
          8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
fprintf(out, "# $%d = number of particles\n\n", 7 + 2 * ctl.nq);

/* Loop over file pairs... */
for (f = 4; f < argc; f += 2) {

  /* Read atmopheric data... */
  if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
    continue;

  /* Check if structs match... */
  if (atm1->np != atm2->np)
    ERRMSG("Different numbers of particles!");

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

  /* Save initial time... */
  if (!init) {
    init = 1;
    t0 = t;
  }

  /* Init... */
  np = 0;
  for (ip = 0; ip < atm1->np; ip++) {
    ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
    for (iq = 0; iq < ctl.nq; iq++)
      aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
  }

  /* Loop over air parcels... */
  for (ip = 0; ip < atm1->np; ip++) {

    /* Check data... */
    if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
      continue;

    /* Check ensemble index... */
    if (ctl.qnt_ens > 0
        && (atm1->q[ctl.qnt_ens][ip] != ens
            || atm2->q[ctl.qnt_ens][ip] != ens))
      continue;

    /* Check spatial range... */
    if (atm1->p[ip] > p0 || atm1->p[ip] < p1
        || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
        || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
      continue;
    if (atm2->p[ip] > p0 || atm2->p[ip] < p1
        || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
        || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
      continue;

    /* Convert coordinates... */
    geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
    geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
    z1 = Z(atm1->p[ip]);
    z2 = Z(atm2->p[ip]);

    /* Calculate absolute transport deviations... */
    ahtd[np] = DIST(x1, x2);
    avtd[np] = z1 - z2;
    for (iq = 0; iq < ctl.nq; iq++)
      aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];

    /* Calculate relative transport deviations... */
    if (f > 4) {

      /* Get trajectory lengths... */
      geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
      lh1[ip] += DIST(x0, x1);
      lv1[ip] += fabs(z1_old[ip] - z1);

      geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
      lh2[ip] += DIST(x0, x2);
      lv2[ip] += fabs(z2_old[ip] - z2);

      /* Get relative transport deviations... */
      if (lh1[ip] + lh2[ip] > 0)
        rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
      if (lv1[ip] + lv2[ip] > 0)
        rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
      for (iq = 0; iq < ctl.nq; iq++)
        rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
          / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
    }

    /* Save positions of air parcels... */
    lon1_old[ip] = atm1->lon[ip];
    lat1_old[ip] = atm1->lat[ip];
    z1_old[ip] = z1;

    lon2_old[ip] = atm2->lon[ip];
    lat2_old[ip] = atm2->lat[ip];
    z2_old[ip] = z2;

    /* Increment air parcel counter... */
    np++;
  }

  /* Get statistics... */
  if (strcasecmp(argv[3], "mean") == 0) {
    ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "stddev") == 0) {
    ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "min") == 0) {
    ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_min(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "max") == 0) {
    ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_max(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "skew") == 0) {
    ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "kurt") == 0) {
    ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "median") == 0) {
    ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_median(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
      rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
    }
  } else if (strcasecmp(argv[3], "absdev") == 0) {
    ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
    rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
    avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
    rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
```

```
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "mad") == 0) {
      ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
      rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
      avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
      rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
        rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
      }
    } else
      ERRMSG("Unknown parameter!");

    /* Write output... */
    fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
            ahtdm, rhtdm, avtdm, rvtdm);
    for (iq = 0; iq < ctl.nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
    }
    fprintf(out, " %d\n", np);
  }

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

  /* Free... */
  free(atm1);
  free(atm2);
  free(lon1_old);
  free(lat1_old);
  free(z1_old);
  free(lh1);
  free(lv1);
  free(lon2_old);
  free(lat2_old);
  free(z2_old);
  free(lh2);
  free(lv2);
  free(ahtd);
  free(avtd);
  free(aqtd);
  free(rhtd);
  free(rvtd);
  free(rqtd);
  free(work);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.4 atm_dist.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  atm_t *atm1, *atm2;

  FILE *out;

  char tstr[LEN];

  double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
    *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
    *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
    t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work;

  int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;

  /* Allocate... */
  ALLOC(atm1, atm_t, 1);
  ALLOC(atm2, atm_t, 1);
  ALLOC(lon1_old, double,
        NP);
  ALLOC(lat1_old, double,
        NP);
  ALLOC(z1_old, double,
        NP);
  ALLOC(lh1, double,
        NP);
  ALLOC(lv1, double,
        NP);
  ALLOC(lon2_old, double,
        NP);
  ALLOC(lat2_old, double,
        NP);
  ALLOC(z2_old, double,
        NP);
  ALLOC(lh2, double,
        NP);
  ALLOC(lv2, double,
        NP);
  ALLOC(ahtd, double,
        NP);
  ALLOC(avtd, double,
        NP);
  ALLOC(aqtd, double,
        NP * NQ);
  ALLOC(rhtd, double,
        NP);
  ALLOC(rvtd, double,
        NP);
  ALLOC(rqtd, double,
        NP * NQ);
  ALLOC(work, double,
        NP);

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atm1a> <atm1b>"
           " [<atm2a> <atm2b> ...]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
  p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
  p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
  lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);

  /* Write info... */
  printf("Write transport deviations: %s\n", argv[2]);

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

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = time difference [s]\n"
          "# $3 = absolute horizontal distance (%s) [km]\n"
          "# $4 = relative horizontal distance (%s) [%%]\n"
          "# $5 = absolute vertical distance (%s) [km]\n"
          "# $6 = relative vertical distance (%s) [%%]\n",
          argv[3], argv[3], argv[3], argv[3]);
  for (iq = 0; iq < ctl.nq; iq++)
    fprintf(out,
            "# $%d = %s absolute difference (%s) [%s]\n"
            "# $%d = %s relative difference (%s) [%%]\n",
            7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
            8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
  fprintf(out, "# $%d = number of particles\n\n", 7 + 2 * ctl.nq);

  /* Loop over file pairs... */
  for (f = 4; f < argc; f += 2) {

    /* Read atmopheric data... */
    if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
      continue;

    /* Check if structs match... */
    if (atm1->np != atm2->np)
      ERRMSG("Different numbers of particles!");

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

    /* Save initial time... */
    if (!init) {
      init = 1;
      t0 = t;
    }

    /* Init... */
    np = 0;
    for (ip = 0; ip < atm1->np; ip++) {
      ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
      for (iq = 0; iq < ctl.nq; iq++)
        aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
    }

    /* Loop over air parcels... */
    for (ip = 0; ip < atm1->np; ip++) {

      /* Check data... */
      if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
        continue;

      /* Check ensemble index... */
      if (ctl.qnt_ens > 0
          && (atm1->q[ctl.qnt_ens][ip] != ens
              || atm2->q[ctl.qnt_ens][ip] != ens))
        continue;

      /* Check spatial range... */
      if (atm1->p[ip] > p0 || atm1->p[ip] < p1
          || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
          || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
        continue;
      if (atm2->p[ip] > p0 || atm2->p[ip] < p1
          || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
          || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
        continue;

      /* Convert coordinates... */
      geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
      geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
      z1 = Z(atm1->p[ip]);
      z2 = Z(atm2->p[ip]);

      /* Calculate absolute transport deviations... */
      ahtd[np] = DIST(x1, x2);
      avtd[np] = z1 - z2;
      for (iq = 0; iq < ctl.nq; iq++)
        aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];

      /* Calculate relative transport deviations... */
      if (f > 4) {

        /* Get trajectory lengths... */
        geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
        lh1[ip] += DIST(x0, x1);
        lv1[ip] += fabs(z1_old[ip] - z1);

        geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
        lh2[ip] += DIST(x0, x2);
        lv2[ip] += fabs(z2_old[ip] - z2);

        /* Get relative transport deviations... */
        if (lh1[ip] + lh2[ip] > 0)
          rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
        if (lv1[ip] + lv2[ip] > 0)
          rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
        for (iq = 0; iq < ctl.nq; iq++)
          rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
            / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
      }

      /* Save positions of air parcels... */
      lon1_old[ip] = atm1->lon[ip];
      lat1_old[ip] = atm1->lat[ip];
      z1_old[ip] = z1;

      lon2_old[ip] = atm2->lon[ip];
      lat2_old[ip] = atm2->lat[ip];
      z2_old[ip] = z2;

      /* Increment air parcel counter... */
      np++;
    }

    /* Get statistics... */
    if (strcasecmp(argv[3], "mean") == 0) {
      ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "stddev") == 0) {
      ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "min") == 0) {
      ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_min(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "max") == 0) {
      ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_max(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "skew") == 0) {
      ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "kurt") == 0) {
      ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "median") == 0) {
      ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_median(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "absdev") == 0) {
      ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
      rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
      avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
      rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
        rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
      }
    } else if (strcasecmp(argv[3], "mad") == 0) {
      ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
      rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
      avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
      rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
```

```
        rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
      }
    } else
      ERRMSG("Unknown parameter!");

    /* Write output... */
    fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
            ahtdm, rhtdm, avtdm, rvtdm);
    for (iq = 0; iq < ctl.nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
    }
    fprintf(out, " %d\n", np);
  }

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

  /* Free... */
  free(atm1);
  free(atm2);
  free(lon1_old);
  free(lat1_old);
  free(z1_old);
  free(lh1);
  free(lv1);
  free(lon2_old);
  free(lat2_old);
  free(z2_old);
  free(lh2);
  free(lv2);
  free(ahtd);
  free(avtd);
  free(aqtd);
  free(rhtd);
  free(rvtd);
  free(rqtd);
  free(work);

  return EXIT_SUCCESS;
}
```

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.

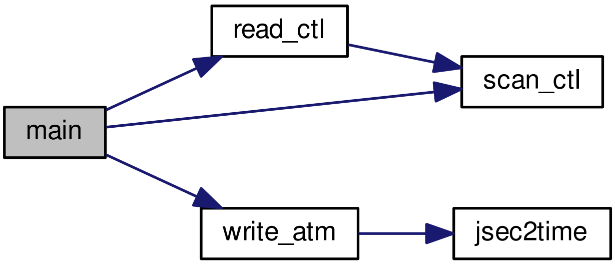
5.5.2 Function Documentation

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

Definition at line 27 of file atm_init.c.

```
                {

  atm_t *atm;

  ctl_t ctl;

  gsl_rng *rng;

  double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
    t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;

  int even, ip, irep, rep;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <atm_out>");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
  t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
  dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
  z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
  dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
  st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
  sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
  slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
  slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
  sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
  ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
  uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
  ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
  ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
  even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
  rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
  m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);

  /* Initialize random number generator... */
  gsl_rng_env_setup();
  rng = gsl_rng_alloc(gsl_rng_default);

  /* Create grid... */
  for (t = t0; t <= t1; t += dt)
    for (z = z0; z <= z1; z += dz)
      for (lon = lon0; lon <= lon1; lon += dlon)
        for (lat = lat0; lat <= lat1; lat += dlat)
          for (irep = 0; irep < rep; irep++) {

            /* Set position... */
            atm->time[atm->np]
              = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
                 + ut * (gsl_rng_uniform(rng) - 0.5));
            atm->p[atm->np]
              = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
                  + uz * (gsl_rng_uniform(rng) - 0.5));
            atm->lon[atm->np]
              = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
                 + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
                 + ulon * (gsl_rng_uniform(rng) - 0.5));
            do {
              atm->lat[atm->np]
                = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
                   + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
                   + ulat * (gsl_rng_uniform(rng) - 0.5));
            } while (even && gsl_rng_uniform(rng) >
                     fabs(cos(atm->lat[atm->np] * M_PI / 180.)));

            /* Set particle counter... */
            if ((++atm->np) >= NP)
              ERRMSG("Too many particles!");
          }

  /* Check number of air parcels... */
  if (atm->np <= 0)
    ERRMSG("Did not create any air parcels!");

  /* Initialize mass... */
  if (ctl.qnt_m >= 0)
    for (ip = 0; ip < atm->np; ip++)
      atm->q[ctl.qnt_m][ip] = m / atm->np;

  /* Save data... */
  write_atm(argv[2], &ctl, atm, t0);

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.6 atm_init.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  atm_t *atm;

  ctl_t ctl;

  gsl_rng *rng;

  double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
    t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;

  int even, ip, irep, rep;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <atm_out>");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
  t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
  dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
  z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
  dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
  st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
  sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
  slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
  slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
  sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
  ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
  uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
  ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
  ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
  even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
  rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
  m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);

  /* Initialize random number generator... */
  gsl_rng_env_setup();
  rng = gsl_rng_alloc(gsl_rng_default);

  /* Create grid... */
  for (t = t0; t <= t1; t += dt)
    for (z = z0; z <= z1; z += dz)
      for (lon = lon0; lon <= lon1; lon += dlon)
        for (lat = lat0; lat <= lat1; lat += dlat)
          for (irep = 0; irep < rep; irep++) {

            /* Set position... */
            atm->time[atm->np]
              = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
                 + ut * (gsl_rng_uniform(rng) - 0.5));
            atm->p[atm->np]
              = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
                  + uz * (gsl_rng_uniform(rng) - 0.5));
            atm->lon[atm->np]
              = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
                 + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
                 + ulon * (gsl_rng_uniform(rng) - 0.5));
            do {
              atm->lat[atm->np]
                = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
                   + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
                   + ulat * (gsl_rng_uniform(rng) - 0.5));
            } while (even && gsl_rng_uniform(rng) >
                     fabs(cos(atm->lat[atm->np] * M_PI / 180.)));

            /* Set particle counter... */
            if ((++atm->np) >= NP)
              ERRMSG("Too many particles!");
          }

  /* Check number of air parcels... */
  if (atm->np <= 0)
    ERRMSG("Did not create any air parcels!");

  /* Initialize mass... */
  if (ctl.qnt_m >= 0)
    for (ip = 0; ip < atm->np; ip++)
      atm->q[ctl.qnt_m][ip] = m / atm->np;

  /* Save data... */
  write_atm(argv[2], &ctl, atm, t0);

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

  return EXIT_SUCCESS;
}
```

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.

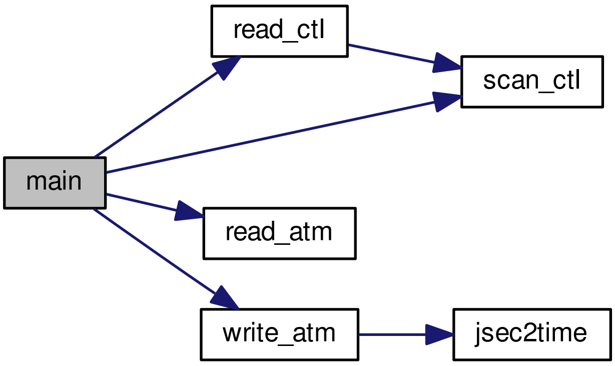
5.7.2 Function Documentation

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

Definition at line 27 of file atm_split.c.

```
                {

  atm_t *atm, *atm2;

  ctl_t ctl;

  gsl_rng *rng;

  double m, mtot = 0, dt, dx, dz, mmax = 0,
    t0, t1, z0, z1, lon0, lon1, lat0, lat1;

  int i, ip, iq, n;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);
  ALLOC(atm2, atm_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
  m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
  dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
  t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
  t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
  dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
  z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
  dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);

  /* Init random number generator... */
  gsl_rng_env_setup();
  rng = gsl_rng_alloc(gsl_rng_default);

  /* Read atmospheric data... */
  if (!read_atm(argv[2], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Get total and maximum mass... */
  if (ctl.qnt_m >= 0)
    for (ip = 0; ip < atm->np; ip++) {
      mtot += atm->q[ctl.qnt_m][ip];
      mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
    }
  if (m > 0)
    mtot = m;

  /* Loop over air parcels... */
  for (i = 0; i < n; i++) {

    /* Select air parcel... */
    if (ctl.qnt_m >= 0)
      do {
        ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
      } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
    else
      ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);

    /* Set time... */
    if (t1 > t0)
      atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
    else
      atm2->time[atm2->np] = atm->time[ip]
        + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);

    /* Set vertical position... */
    if (z1 > z0)
      atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
    else
      atm2->p[atm2->np] = atm->p[ip]
        + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);

    /* Set horizontal position... */
    if (lon1 > lon0 && lat1 > lat0) {
      atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
      atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
    } else {
      atm2->lon[atm2->np] = atm->lon[ip]
        + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
      atm2->lat[atm2->np] = atm->lat[ip]
        + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
    }

    /* Copy quantities... */
    for (iq = 0; iq < ctl.nq; iq++)
      atm2->q[iq][atm2->np] = atm->q[iq][ip];

    /* Adjust mass... */
    if (ctl.qnt_m >= 0)
      atm2->q[ctl.qnt_m][atm2->np] = mtot / n;

    /* Increment particle counter... */
    if ((++atm2->np) >= NP)
      ERRMSG("Too many air parcels!");
  }

  /* Save data and close file... */
  write_atm(argv[3], &ctl, atm2, atm->time[0]);

  /* Free... */
  free(atm);
  free(atm2);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.8 atm_split.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  atm_t *atm, *atm2;

  ctl_t ctl;

  gsl_rng *rng;

  double m, mtot = 0, dt, dx, dz, mmax = 0,
    t0, t1, z0, z1, lon0, lon1, lat0, lat1;

  int i, ip, iq, n;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);
  ALLOC(atm2, atm_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
  m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
  dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
  t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
  t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
  dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
  z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
  dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);

  /* Init random number generator... */
  gsl_rng_env_setup();
  rng = gsl_rng_alloc(gsl_rng_default);

  /* Read atmospheric data... */
  if (!read_atm(argv[2], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Get total and maximum mass... */
  if (ctl.qnt_m >= 0)
    for (ip = 0; ip < atm->np; ip++) {
      mtot += atm->q[ctl.qnt_m][ip];
      mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
    }
  if (m > 0)
    mtot = m;

  /* Loop over air parcels... */
  for (i = 0; i < n; i++) {

    /* Select air parcel... */
    if (ctl.qnt_m >= 0)
      do {
        ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
      } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
    else
      ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);

    /* Set time... */
    if (t1 > t0)
      atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
    else
      atm2->time[atm2->np] = atm->time[ip]
        + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);

    /* Set vertical position... */
    if (z1 > z0)
      atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
    else
      atm2->p[atm2->np] = atm->p[ip]
        + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);

    /* Set horizontal position... */
    if (lon1 > lon0 && lat1 > lat0) {
      atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
      atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
    } else {
      atm2->lon[atm2->np] = atm->lon[ip]
        + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
      atm2->lat[atm2->np] = atm->lat[ip]
        + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
    }

    /* Copy quantities... */
    for (iq = 0; iq < ctl.nq; iq++)
      atm2->q[iq][atm2->np] = atm->q[iq][ip];

    /* Adjust mass... */
    if (ctl.qnt_m >= 0)
      atm2->q[ctl.qnt_m][atm2->np] = mtot / n;

    /* Increment particle counter... */
    if ((++atm2->np) >= NP)
      ERRMSG("Too many air parcels!");
  }

  /* Save data and close file... */
  write_atm(argv[3], &ctl, atm2, atm->time[0]);

  /* Free... */
  free(atm);
  free(atm2);

  return EXIT_SUCCESS;
}
```

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

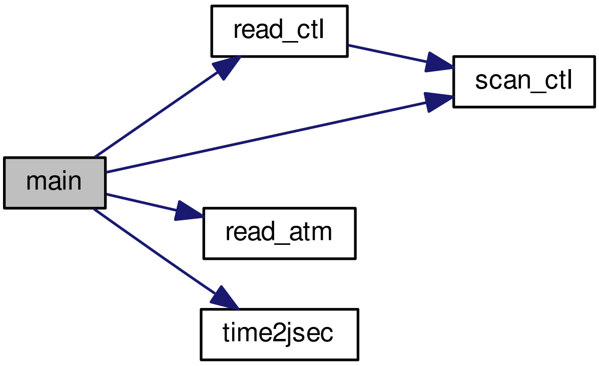
5.9.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 27 of file atm_stat.c.

```
              {

ctl_t ctl;

atm_t *atm, *atm_filt;

FILE *out;

char tstr[LEN];

double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
  t, t0, qm[NQ], *work, zm, *zs;

int ens, f, init = 0, ip, iq, year, mon, day, hour, min;

/* Allocate... */
ALLOC(atm, atm_t, 1);
ALLOC(atm_filt, atm_t, 1);
ALLOC(work, double,
      NP);
ALLOC(zs, double,
      NP);

/* Check arguments... */
if (argc < 4)
  ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");

/* Read control parameters... */
read_ctl(argv[1], argc, argv, &ctl);
ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);

/* Write info... */
printf("Write air parcel statistics: %s\n", argv[2]);

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

/* Write header... */
fprintf(out,
        "# $1 = time [s]\n"
        "# $2 = time difference [s]\n"
```

```
          "# $3 = altitude (%s) [km]\n"
          "# $4 = longitude (%s) [deg]\n"
          "# $5 = latitude (%s) [deg]\n", argv[3], argv[3], argv[3]);
  for (iq = 0; iq < ctl.nq; iq++)
    fprintf(out, "# $%d = %s (%s) [%s]\n", iq + 6,
            ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
  fprintf(out, "# $%d = number of particles\n\n", ctl.nq + 6);

  /* Loop over files... */
  for (f = 4; f < argc; f++) {

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

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

    /* Save intial time... */
    if (!init) {
      init = 1;
      t0 = t;
    }

    /* Filter data... */
    atm_filt->np = 0;
    for (ip = 0; ip < atm->np; ip++) {

      /* Check time... */
      if (!gsl_finite(atm->time[ip]))
        continue;

      /* Check ensemble index... */
      if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
        continue;

      /* Check spatial range... */
      if (atm->p[ip] > p0 || atm->p[ip] < p1
          || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
          || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
        continue;

      /* Save data... */
      atm_filt->time[atm_filt->np] = atm->time[ip];
      atm_filt->p[atm_filt->np] = atm->p[ip];
      atm_filt->lon[atm_filt->np] = atm->lon[ip];
      atm_filt->lat[atm_filt->np] = atm->lat[ip];
      for (iq = 0; iq < ctl.nq; iq++)
        atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
      atm_filt->np++;
    }

    /* Get heights... */
    for (ip = 0; ip < atm_filt->np; ip++)
      zs[ip] = Z(atm_filt->p[ip]);

    /* Get statistics... */
    if (strcasecmp(argv[3], "mean") == 0) {
      zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "stddev") == 0) {
      zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "min") == 0) {
      zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "max") == 0) {
      zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "skew") == 0) {
      zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "kurt") == 0) {
      zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] =
          gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "median") == 0) {
      zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "absdev") == 0) {
      zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "mad") == 0) {
      zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
      lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
      latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] =
          gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
    } else
      ERRMSG("Unknown parameter!");

    /* Write data... */
    fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
    for (iq = 0; iq < ctl.nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], qm[iq]);
    }
    fprintf(out, " %d\n", atm_filt->np);
  }

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

  /* Free... */
  free(atm);
  free(atm_filt);
  free(work);
  free(zs);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.10 atm_stat.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  atm_t *atm, *atm_filt;

  FILE *out;

  char tstr[LEN];

  double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
    t, t0, qm[NQ], *work, zm, *zs;

  int ens, f, init = 0, ip, iq, year, mon, day, hour, min;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);
  ALLOC(atm_filt, atm_t, 1);
  ALLOC(work, double,
        NP);
  ALLOC(zs, double,
        NP);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
  p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
  p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
  lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);

  /* Write info... */
  printf("Write air parcel statistics: %s\n", argv[2]);

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

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = time difference [s]\n"
          "# $3 = altitude (%s) [km]\n"
          "# $4 = longitude (%s) [deg]\n"
          "# $5 = latitude (%s) [deg]\n", argv[3], argv[3], argv[3]);
  for (iq = 0; iq < ctl.nq; iq++)
    fprintf(out, "# $%d = %s (%s) [%s]\n", iq + 6,
            ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
  fprintf(out, "# $%d = number of particles\n\n", ctl.nq + 6);

  /* Loop over files... */
  for (f = 4; f < argc; f++) {

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

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

    /* Save intial time... */
    if (!init) {
      init = 1;
      t0 = t;
    }

    /* Filter data... */
    atm_filt->np = 0;
    for (ip = 0; ip < atm->np; ip++) {

      /* Check time... */
      if (!gsl_finite(atm->time[ip]))
        continue;

      /* Check ensemble index... */
      if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
        continue;

      /* Check spatial range... */
      if (atm->p[ip] > p0 || atm->p[ip] < p1
          || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
          || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
        continue;

      /* Save data... */
      atm_filt->time[atm_filt->np] = atm->time[ip];
      atm_filt->p[atm_filt->np] = atm->p[ip];
      atm_filt->lon[atm_filt->np] = atm->lon[ip];
      atm_filt->lat[atm_filt->np] = atm->lat[ip];
      for (iq = 0; iq < ctl.nq; iq++)
        atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
      atm_filt->np++;
    }

    /* Get heights... */
    for (ip = 0; ip < atm_filt->np; ip++)
      zs[ip] = Z(atm_filt->p[ip]);

    /* Get statistics... */
    if (strcasecmp(argv[3], "mean") == 0) {
      zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "stddev") == 0) {
      zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "min") == 0) {
      zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "max") == 0) {
      zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "skew") == 0) {
      zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "kurt") == 0) {
      zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] =
          gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "median") == 0) {
      zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "absdev") == 0) {
      zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
      lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
      latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
    } else if (strcasecmp(argv[3], "mad") == 0) {
      zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
      lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
      latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
      for (iq = 0; iq < ctl.nq; iq++)
        qm[iq] =
          gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
    } else
      ERRMSG("Unknown parameter!");

    /* Write data... */
    fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
    for (iq = 0; iq < ctl.nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], qm[iq]);
    }
    fprintf(out, " %d\n", atm_filt->np);
  }

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

  /* Free... */
  free(atm);
  free(atm_filt);
  free(work);
  free(zs);

  return EXIT_SUCCESS;
}
```

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.

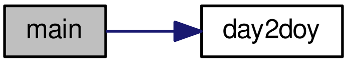
5.11.2 Function Documentation

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

Definition at line 27 of file day2doy.c.

```
                {

  int day, doy, mon, year;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <year> <mon> <day>");

  /* Read arguments... */
  year = atoi(argv[1]);
  mon = atoi(argv[2]);
  day = atoi(argv[3]);

  /* Convert... */
  day2doy(year, mon, day, &doy);
  printf("%d %d\n", year, doy);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.12 day2doy.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  int day, doy, mon, year;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <year> <mon> <day>");

  /* Read arguments... */
  year = atoi(argv[1]);
  mon = atoi(argv[2]);
  day = atoi(argv[3]);

  /* Convert... */
  day2doy(year, mon, day, &doy);
  printf("%d %d\n", year, doy);

  return EXIT_SUCCESS;
}
```

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.

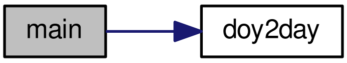
5.13.2 Function Documentation

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 
```

```
  /* Read arguments... */
  year = atoi(argv[1]);
  doy = atoi(argv[2]);

  /* Convert... */
  doy2day(year, doy, &mon, &day);
  printf("%d %d %d\n", year, mon, day);

  return EXIT_SUCCESS;
}
```

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

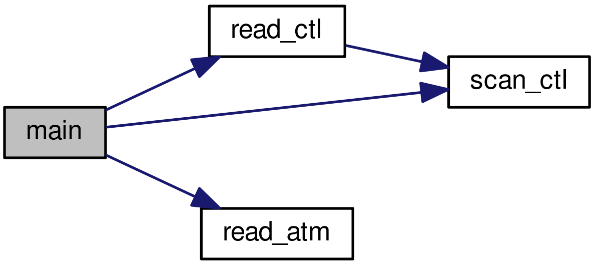
5.15.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 27 of file extract.c.

```
                {

  ctl_t ctl;

  atm_t *atm;

  FILE *out;

  int f, ip, iq;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <trajec.tab> <atm1> [<atm2> ...]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "", NULL);

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

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

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
  for (iq = 0; iq < ctl.nq; iq++)
```

```
    fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
            ctl.qnt_unit[iq]);
  fprintf(out, "\n");

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

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

    /* Check air parcel index... */
    if (ip > atm->np)
      ERRMSG("Air parcel index out of range!");

    /* Write data... */
    fprintf(out, "%.2f %g %g %g", atm->time[ip],
            Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
    for (iq = 0; iq < ctl.nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.16 extract.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/
```

```c

#include "libtrac.h"

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

  ctl_t ctl;

  atm_t *atm;

  FILE *out;

  int f, ip, iq;

  /* Allocate... */
  ALLOC(atm, atm_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <trajec.tab> <atm1> [<atm2> ...]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "", NULL);

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

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

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
  for (iq = 0; iq < ctl.nq; iq++)
    fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
            ctl.qnt_unit[iq]);
  fprintf(out, "\n");

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

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

    /* Check air parcel index... */
    if (ip > atm->np)
      ERRMSG("Air parcel index out of range!");

    /* Write data... */
    fprintf(out, "%.2f %g %g %g", atm->time[ip],
            Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
    for (iq = 0; iq < ctl.nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

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

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

  return EXIT_SUCCESS;
}
```

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.

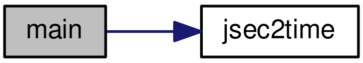
5.17.2 Function Documentation

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

Definition at line 27 of file jsec2time.c.

```
                 {

  double jsec, remain;

  int day, hour, min, mon, sec, year;

  /* Check arguments... */
  if (argc < 2)
    ERRMSG("Give parameters: <jsec>");

  /* Read arguments... */
  jsec = atof(argv[1]);

  /* Convert time... */
  jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
  printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.18 jsec2time.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

```

```
#include "libtrac.h"

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

  double jsec, remain;

  int day, hour, min, mon, sec, year;

  /* Check arguments... */
  if (argc < 2)
    ERRMSG("Give parameters: <jsec>");

  /* Read arguments... */
  jsec = atof(argv[1]);

  /* Convert time... */
  jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
  printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);

  return EXIT_SUCCESS;
}
```

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.

```
               {

  double radius;

  radius = NORM(x);
  *lat = asin(x[2] / radius) * 180 / M_PI;
  *lon = atan2(x[1], x[0]) * 180 / M_PI;
  *z = radius - RE;
}
```

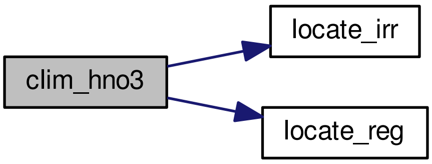
5.19.2.2 double clim_hno3 (double *t,* double *lat,* double *p*)

Climatology of HNO3 volume mixing ratios.

Definition at line 45 of file libtrac.c.

```
            {

  static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
    9072000.00, 11664000.00, 14342400.00,
    16934400.00, 19612800.00, 22291200.00,
    24883200.00, 27561600.00, 30153600.00
  };

  static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
    5, 15, 25, 35, 45, 55, 65, 75, 85
  };

  static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
    31.6228, 46.4159, 68.1292, 100, 146.78
  };

  static double hno3[12][18][10] = {
    {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
     {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
     {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
     {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
     {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
     {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
     {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
     {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
     {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
     {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
     {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
     {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
     {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
     {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
     {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
     {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
     {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
     {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
    {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
     {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
     {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
     {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
     {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
     {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
     {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
     {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
     {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
     {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
     {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
     {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
     {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
     {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
     {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
     {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
     {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
     {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
    {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
     {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
     {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
     {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
     {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
     {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
     {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
     {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
     {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
     {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
     {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
     {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
     {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
     {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
     {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
     {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
     {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
     {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
    {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
     {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
     {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
     {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
     {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
     {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
     {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
     {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
     {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
     {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
     {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
     {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
     {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
     {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
     {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
     {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
     {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
     {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
    {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
     {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
     {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
     {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
     {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
     {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
     {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
     {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
     {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
     {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
     {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
     {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
     {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
     {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
     {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
     {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
     {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
     {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
    {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
     {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
     {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
     {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
     {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
     {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
     {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
     {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
     {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
     {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
     {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
     {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
     {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
     {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
     {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
     {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
     {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
     {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
    {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
     {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
     {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
     {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
     {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
     {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
     {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
     {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
     {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
     {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
     {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
     {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
     {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
     {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
     {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
     {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
     {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
     {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
    {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
     {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
     {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
     {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
     {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
     {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
     {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
     {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
     {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
     {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
     {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
    {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
    {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
    {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
    {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
    {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
    {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
    {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
    {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
    {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
    {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
    {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
    {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
  };

  double aux00, aux01, aux10, aux11, sec;

  int ilat, ip, isec;

  /* Get seconds since begin of year... */
  sec = fmod(t, 365.25 * 86400.);

  /* Get indices... */
  isec = locate_irr(secs, 12, sec);
  ilat = locate_reg(lats, 18, lat);
  ip = locate_irr(ps, 10, p);

  /* Interpolate... */
  aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
              ps[ip + 1], hno3[isec][ilat][ip + 1], p);
  aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
              ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
  aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
              ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
  aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
              ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
  aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
  return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
}
```

Here is the call graph for this function:



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

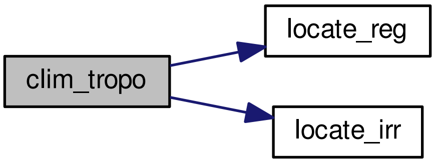
Climatology of tropopause pressure.

Definition at line 311 of file libtrac.c.

```
             {

  static double doys[12]
  = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };

  static double lats[73]
    = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
    -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
    -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
    -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
    15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
    45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
    75, 77.5, 80, 82.5, 85, 87.5, 90
  };
```

```

static double tps[12][73]
  = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
       297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
       175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
       99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
       98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
       152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
       277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
       275.3, 275.6, 275.4, 274.1, 273.5},
{337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
 300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
 150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
 98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
 98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
 220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
 284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
 287.5, 286.2, 285.8},
{335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
 297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
 161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
 100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
 99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
 186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
 279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
 304.3, 304.9, 306, 306.6, 306.2, 306},
{306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
 290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
 195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
 148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
 263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
 315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
{266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
 101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
 102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
 273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
 325.3, 325.8, 325.8},
{220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
 308.5, 312.2, 313.1, 313.3},
{187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
{166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
 230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
 278.2, 282.6, 287.4, 290.9, 292.5, 293},
{171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
 110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
{191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
 112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
 305.1},
{241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
   102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
   109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
   241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
   286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
  {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
   284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
   175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
   100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
   100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
   186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
   280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
   281.7, 281.1, 281.2}
  };

  double doy, p0, p1;

  int imon, ilat;

  /* Get day of year... */
  doy = fmod(t / 86400., 365.25);
  while (doy < 0)
    doy += 365.25;

  /* Get indices... */
  ilat = locate_reg(lats, 73, lat);
  imon = locate_irr(doys, 12, doy);

  /* Interpolate... */
  p0 = LIN(lats[ilat], tps[imon][ilat],
           lats[ilat + 1], tps[imon][ilat + 1], lat);
  p1 = LIN(lats[ilat], tps[imon + 1][ilat],
           lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
  return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
}
```

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.

```
              {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

  /* Get day of year... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
    *doy = d0l[mon - 1] + day - 1;
  else
    *doy = d0[mon - 1] + day - 1;
}
```

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.

```
                  {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
  int i;

  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d01[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d01[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}
```

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.

```
                   {

  double radius;

  radius = z + RE;
  x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
  x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
  x[2] = radius * sin(lat / 180 * M_PI);
}
```

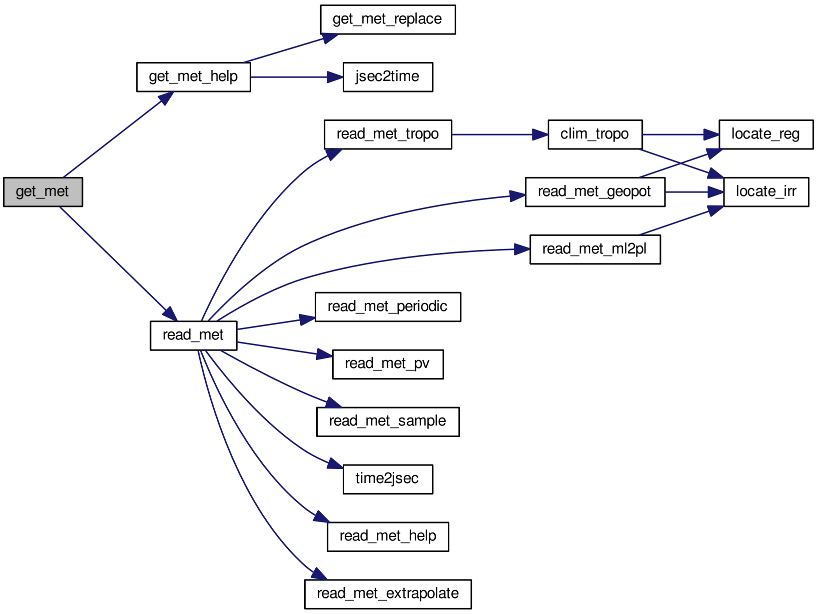
5.19.2.7 void get_met (ctl_t ∗ *ctl,* char ∗ *metbase,* double *t,* met_t ∗∗ *met0,* met_t ∗∗ *met1*)

Get meteorological data for given timestep.

Definition at line 513 of file libtrac.c.

```
                        {

  static int init, ip, ix, iy;

  met_t *mets;

  char filename[LEN];

  /* Init... */
  if (t == ctl->t_start || !init) {
    init = 1;

    get_met_help(t, -1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met0))
      ERRMSG("Cannot open file!");

    get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met1))
      ERRMSG("Cannot open file!");
  }

  /* Read new data for forward trajectories... */
  if (t > (*met1)->time && ctl->direction == 1) {
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;
    get_met_help(t, 1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met1))
      ERRMSG("Cannot open file!");
  }

  /* Read new data for backward trajectories... */
  if (t < (*met0)->time && ctl->direction == -1) {
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;
    get_met_help(t, -1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met0))
      ERRMSG("Cannot open file!");
  }

  /* Check that grids are consistent... */
  if ((*met0)->nx != (*met1)->nx
      || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
    ERRMSG("Meteo grid dimensions do not match!");
  for (ix = 0; ix < (*met0)->nx; ix++)
    if ((*met0)->lon[ix] != (*met1)->lon[ix])
      ERRMSG("Meteo grid longitudes do not match!");
  for (iy = 0; iy < (*met0)->ny; iy++)
    if ((*met0)->lat[iy] != (*met1)->lat[iy])
      ERRMSG("Meteo grid latitudes do not match!");
  for (ip = 0; ip < (*met0)->np; ip++)
    if ((*met0)->p[ip] != (*met1)->p[ip])
      ERRMSG("Meteo grid pressure levels do not match!");
}
```

Here is the call graph for this function:



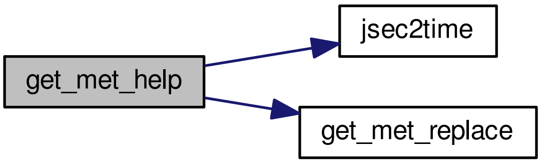
5.19.2.8 void get_met_help (double *t,* int *direct,* char ∗ *metbase,* double *dt_met,* char ∗ *filename*)

Get meteorological data for timestep.

Definition at line 576 of file libtrac.c.

```
                  {

  char repl[LEN];

  double t6, r;

  int year, mon, day, hour, min, sec;

  /* Round time to fixed intervals... */
  if (direct == -1)
    t6 = floor(t / dt_met) * dt_met;
  else
    t6 = ceil(t / dt_met) * dt_met;

  /* Decode time... */
  jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Set filename... */
  sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
  sprintf(repl, "%d", year);
  get_met_replace(filename, "YYYY", repl);
  sprintf(repl, "%02d", mon);
  get_met_replace(filename, "MM", repl);
  sprintf(repl, "%02d", day);
  get_met_replace(filename, "DD", repl);
  sprintf(repl, "%02d", hour);
  get_met_replace(filename, "HH", repl);
}
```

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.

```
              {

  char buffer[LEN], *ch;

  int i;

  /* Iterate... */
  for (i = 0; i < 3; i++) {

    /* Replace substring... */
    if (!(ch = strstr(orig, search)))
      return;
    strncpy(buffer, orig, (size_t) (ch - orig));
    buffer[ch - orig] = 0;
    sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
    orig[0] = 0;
    strcpy(orig, buffer);
  }
}
```

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.

```
                {

  double aux00, aux01, aux10, aux11;

  /* Set variables... */
  aux00 = array[ix][iy];
  aux01 = array[ix][iy + 1];
  aux10 = array[ix + 1][iy];
  aux11 = array[ix + 1][iy + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}
```

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.

```
                {

  double aux00, aux01, aux10, aux11;

  /* Interpolate vertically... */
  aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
    + array[ix][iy][ip + 1];
  aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
    + array[ix][iy + 1][ip + 1];
  aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
    + array[ix + 1][iy][ip + 1];
  aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
    + array[ix + 1][iy + 1][ip + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}
```

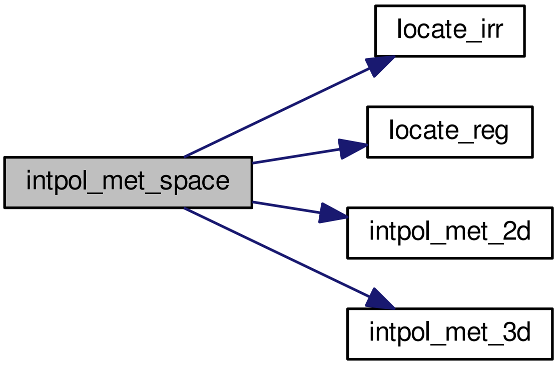
5.19.2.12 void intpol_met_space (met_t * met, double p, double lon, double lat, double * ps, double * pt, double * z, double * t, double * u, double * v, double * w, double * pv, double * h2o, double * o3)

Spatial interpolation of meteorological data.

Definition at line 691 of file libtrac.c.

```
               {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (met->lon[met->nx - 1] > 180 && lon < 0)
    lon += 360;

  /* Get indices... */
  ip = locate_irr(met->p, met->np, p);
  ix = locate_reg(met->lon, met->nx, lon);
  iy = locate_reg(met->lat, met->ny, lat);

  /* Get weights... */
  wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
  wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
  wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);

  /* Interpolate... */
  if (ps != NULL)
    intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
  if (pt != NULL)
    intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
  if (z != NULL)
    intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
  if (t != NULL)
    intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
  if (u != NULL)
    intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
  if (v != NULL)
    intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
  if (w != NULL)
    intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
  if (pv != NULL)
    intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
  if (h2o != NULL)
    intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
  if (o3 != NULL)
    intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
}
```

Here is the call graph for this function:



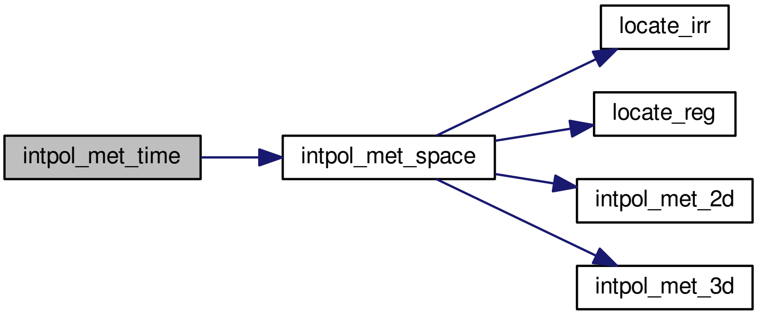
5.19.2.13 void intpol_met_time (met_t * *met0*, met_t * *met1*, double *ts*, double *p*, double *lon*, double *lat*, double * *ps*, double * *pt*, double * *z*, double * *t*, double * *u*, double * *v*, double * *w*, double * *pv*, double * *h2o*, double * *o3*)

Temporal interpolation of meteorological data.

Definition at line 750 of file libtrac.c.

```
                {

  double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
    v0, v1, w0, w1, wt, z0, z1;

  /* Spatial interpolation... */
  intpol_met_space(met0, p, lon, lat,
                   ps == NULL ? NULL : &ps0,
                   pt == NULL ? NULL : &pt0,
                   z == NULL ? NULL : &z0,
                   t == NULL ? NULL : &t0,
                   u == NULL ? NULL : &u0,
                   v == NULL ? NULL : &v0,
                   w == NULL ? NULL : &w0,
                   pv == NULL ? NULL : &pv0,
                   h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
  intpol_met_space(met1, p, lon, lat,
                   ps == NULL ? NULL : &ps1,
                   pt == NULL ? NULL : &pt1,
                   z == NULL ? NULL : &z1,
                   t == NULL ? NULL : &t1,
                   u == NULL ? NULL : &u1,
                   v == NULL ? NULL : &v1,
                   w == NULL ? NULL : &w1,
                   pv == NULL ? NULL : &pv1,
                   h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);

  /* Get weighting factor... */
  wt = (met1->time - ts) / (met1->time - met0->time);

  /* Interpolate... */
  if (ps != NULL)
    *ps = wt * (ps0 - ps1) + ps1;
  if (pt != NULL)
    *pt = wt * (pt0 - pt1) + pt1;
  if (z != NULL)
    *z = wt * (z0 - z1) + z1;
  if (t != NULL)
    *t = wt * (t0 - t1) + t1;
  if (u != NULL)
    *u = wt * (u0 - u1) + u1;
  if (v != NULL)
    *v = wt * (v0 - v1) + v1;
  if (w != NULL)
    *w = wt * (w0 - w1) + w1;
  if (pv != NULL)
    *pv = wt * (pv0 - pv1) + pv1;
  if (h2o != NULL)
    *h2o = wt * (h2o0 - h2o1) + h2o1;
  if (o3 != NULL)
    *o3 = wt * (o30 - o31) + o31;
}
```

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.

```
                  {

  struct tm t0, *t1;

  time_t jsec0;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}
```

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.

```
            {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}
```

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.

```
            {

  int i;

  /* Calculate index... */
  i = (int) ((x - xx[0]) / (xx[1] - xx[0]));

  /* Check range... */
  if (i < 0)
    i = 0;
  else if (i >= n - 2)
    i = n - 2;

  return i;
}
```

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.

```
         {

  FILE *in;

  char line[LEN], *tok;

  double t0;

  int dimid, ip, iq, ncid, varid;

  size_t nparts;

  /* Init... */
  atm->np = 0;

  /* Write info... */
  printf("Read atmospheric data: %s\n", filename);

  /* Read ASCII data... */
  if (ctl->atm_type == 0) {

   /* Open file... */
   if (!(in = fopen(filename, "r")))
    return 0;

   /* Read line... */
   while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    for (iq = 0; iq < ctl->nq; iq++)
     TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

    /* Convert altitude to pressure... */
    atm->p[atm->np] = P(atm->p[atm->np]);

    /* Increment data point counter... */
    if ((++atm->np) > NP)
     ERRMSG("Too many data points!");
   }

   /* Close file... */
   fclose(in);
  }

  /* Read binary data... */
  else if (ctl->atm_type == 1) {

   /* Open file... */
   if (!(in = fopen(filename, "r")))
    return 0;

   /* Read data... */
   FREAD(&atm->np, int, 1, in);
   FREAD(atm->time, double,
       (size_t) atm->np,
      in);
   FREAD(atm->p, double,
       (size_t) atm->np,
      in);
   FREAD(atm->lon, double,
       (size_t) atm->np,
      in);
   FREAD(atm->lat, double,
       (size_t) atm->np,
      in);
   for (iq = 0; iq < ctl->nq; iq++)
    FREAD(atm->q[iq], double,
        (size_t) atm->np,
       in);

   /* Close file... */
   fclose(in);
  }
  /* Read netCDF data... */
  else if (ctl->atm_type == 2) {

   /* Open file... */
   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
    return 0;

   /* Get dimensions... */
   NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
   NC(nc_inq_dimlen(ncid, dimid, &nparts));
   atm->np = (int) nparts;
   if (atm->np > NP)
    ERRMSG("Too many particles!");

   /* Get time... */
   NC(nc_inq_varid(ncid, "time", &varid));
   NC(nc_get_var_double(ncid, varid, &t0));
   for (ip = 0; ip < atm->np; ip++)
    atm->time[ip] = t0;

   /* Read geolocations... */
   NC(nc_inq_varid(ncid, "PRESS", &varid));
   NC(nc_get_var_double(ncid, varid, atm->p));
   NC(nc_inq_varid(ncid, "LON", &varid));
   NC(nc_get_var_double(ncid, varid, atm->lon));
   NC(nc_inq_varid(ncid, "LAT", &varid));
   NC(nc_get_var_double(ncid, varid, atm->lat));

   /* Read variables... */
   if (ctl->qnt_p >= 0)
    if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
   if (ctl->qnt_t >= 0)
    if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
   if (ctl->qnt_u >= 0)
    if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
   if (ctl->qnt_v >= 0)
    if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
   if (ctl->qnt_w >= 0)
    if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
   if (ctl->qnt_h2o >= 0)
    if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
   if (ctl->qnt_o3 >= 0)
    if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
   if (ctl->qnt_theta >= 0)
    if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
   if (ctl->qnt_pv >= 0)
    if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
     NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));

   /* Check data... */
   for (ip = 0; ip < atm->np; ip++)
    if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
        || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
        || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
        || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
        || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
     atm->time[ip] = GSL_NAN;
     atm->p[ip] = GSL_NAN;
     atm->lon[ip] = GSL_NAN;
     atm->lat[ip] = GSL_NAN;
     for (iq = 0; iq < ctl->nq; iq++)
      atm->q[iq][ip] = GSL_NAN;
    } else {
     if (ctl->qnt_h2o >= 0)
      atm->q[ctl->qnt_h2o][ip] *= 1.608;
     if (ctl->qnt_pv >= 0)
      atm->q[ctl->qnt_pv][ip] *= 1e6;
     if (atm->lon[ip] > 180)
      atm->lon[ip] -= 360;
    }

   /* Close file... */
   NC(nc_close(ncid));
  }

  /* Error... */
  else
   ERRMSG("Atmospheric data type not supported!");

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

  /* Return success... */
  return 1;
}
```

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

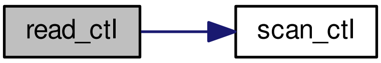
Read control parameters.

Definition at line 1087 of file libtrac.c.

```
               {

  int ip, iq;

  /* Write info... */
  printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
         "(executable: %s | compiled: %s, %s)\n\n",
         argv[0], __DATE__, __TIME__);

  /* Initialize quantity indices... */
  ctl->qnt_ens = -1;
  ctl->qnt_m = -1;
  ctl->qnt_r = -1;
  ctl->qnt_rho = -1;
  ctl->qnt_ps = -1;
  ctl->qnt_pt = -1;
  ctl->qnt_z = -1;
  ctl->qnt_p = -1;
  ctl->qnt_t = -1;
  ctl->qnt_u = -1;
  ctl->qnt_v = -1;
  ctl->qnt_w = -1;
  ctl->qnt_h2o = -1;
  ctl->qnt_o3 = -1;
  ctl->qnt_theta = -1;
  ctl->qnt_vh = -1;
  ctl->qnt_vz = -1;
  ctl->qnt_pv = -1;
  ctl->qnt_tice = -1;
  ctl->qnt_tsts = -1;
  ctl->qnt_tnat = -1;
  ctl->qnt_stat = -1;

  /* Read quantities... */
  ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
  if (ctl->nq > NQ)
    ERRMSG("Too many quantities!");
  for (iq = 0; iq < ctl->nq; iq++) {

    /* Read quantity name and format... */
    scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
    scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
             ctl->qnt_format[iq]);

    /* Try to identify quantity... */
    if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
      ctl->qnt_ens = iq;
      sprintf(ctl->qnt_unit[iq], "-");
    } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
      ctl->qnt_m = iq;
      sprintf(ctl->qnt_unit[iq], "kg");
    } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
      ctl->qnt_r = iq;
      sprintf(ctl->qnt_unit[iq], "m");
    } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
      ctl->qnt_rho = iq;
      sprintf(ctl->qnt_unit[iq], "kg/m^3");
    } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
      ctl->qnt_ps = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
      ctl->qnt_pt = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
      ctl->qnt_z = iq;
      sprintf(ctl->qnt_unit[iq], "km");
    } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
      ctl->qnt_p = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
      ctl->qnt_t = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
      ctl->qnt_u = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
      ctl->qnt_v = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
      ctl->qnt_w = iq;
      sprintf(ctl->qnt_unit[iq], "hPa/s");
    } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
      ctl->qnt_h2o = iq;
      sprintf(ctl->qnt_unit[iq], "1");
    } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
      ctl->qnt_o3 = iq;
      sprintf(ctl->qnt_unit[iq], "1");
    } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
      ctl->qnt_theta = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
      ctl->qnt_vh = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
      ctl->qnt_vz = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
      ctl->qnt_pv = iq;
      sprintf(ctl->qnt_unit[iq], "PVU");
    } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
      ctl->qnt_tice = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
      ctl->qnt_tsts = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
      ctl->qnt_tnat = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
      ctl->qnt_stat = iq;
      sprintf(ctl->qnt_unit[iq], "-");
    } else
      scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
  }

  /* Time steps of simulation... */
  ctl->direction =
    (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
  if (ctl->direction != -1 && ctl->direction != 1)
    ERRMSG("Set DIRECTION to -1 or 1!");
  ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
  ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);

  /* Meteorological data... */
  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
  ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
  ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
  ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
  ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
  ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
  ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
  ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
  if (ctl->met_np > EP)
    ERRMSG("Too many levels!");
  for (ip = 0; ip < ctl->met_np; ip++)
    ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
  ctl->met_tropo
    = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
  scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
  ctl->met_dt_out =
    scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);

  /* Isosurface parameters... */
  ctl->isosurf
    = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

  /* Diffusion parameters... */
  ctl->turb_dx_trop
    = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
  ctl->turb_dx_strat
```

```
    = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
  ctl->turb_dz_trop
    = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
  ctl->turb_dz_strat
    = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
  ctl->turb_mesox =
    scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
  ctl->turb_mesoz =
    scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);

  /* Mass and life time... */
  ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
  ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
  ctl->tdec_strat =
    scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);

  /* PSC analysis... */
  ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
  ctl->psc_hno3 =
    scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);

  /* Output of atmospheric data... */
  scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
  scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
  ctl->atm_dt_out =
    scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
  ctl->atm_filter =
    (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
  ctl->atm_type =
    (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);

  /* Output of CSI data... */
  scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
  ctl->csi_dt_out =
    scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
  scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
  ctl->csi_obsmin =
    scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
  ctl->csi_modmin =
    scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
  ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
  ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
  ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
  ctl->csi_lon0 =
    scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
  ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
  ctl->csi_nx =
    (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
  ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
  ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
  ctl->csi_ny =
    (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);

  /* Output of ensemble data... */
  scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);

  /* Output of grid data... */
  scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
           ctl->grid_basename);
  scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
  ctl->grid_dt_out =
    scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
  ctl->grid_sparse =
    (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
  ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
  ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
  ctl->grid_nz =
    (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
  ctl->grid_lon0 =
    scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
  ctl->grid_lon1 =
    scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
  ctl->grid_nx =
    (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
  ctl->grid_lat0 =
    scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
  ctl->grid_lat1 =
    scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
  ctl->grid_ny =
    (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);

  /* Output of profile data... */
  scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
           ctl->prof_basename);
  scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
  ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
  ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
  ctl->prof_nz =
    (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
  ctl->prof_lon0 =
    scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
  ctl->prof_lon1 =
    scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
  ctl->prof_nx =
    (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
  ctl->prof_lat0 =
    scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
  ctl->prof_lat1 =
    scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
  ctl->prof_ny =
    (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);

  /* Output of station data... */
  scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
           ctl->stat_basename);
  ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
  ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
  ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
}
```

Here is the call graph for this function:



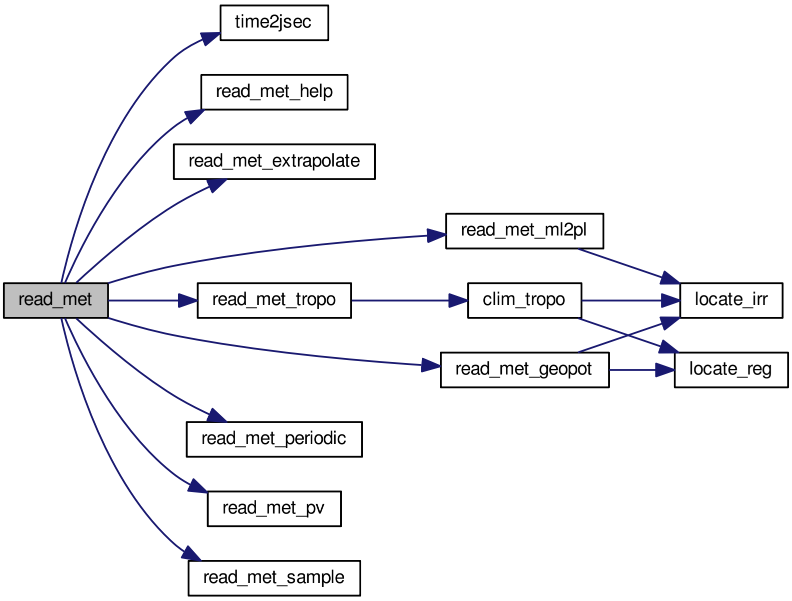
5.19.2.19 int read_met (ctl_t * *ctl,* char * *filename,* met_t * *met*)

Read meteorological data file.

Definition at line 1355 of file libtrac.c.

```
                 {

  char cmd[2 * LEN], levname[LEN], tstr[10];

  float help[EX * EY];

  int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;

  size_t np, nx, ny;

  /* Write info... */
  printf("Read meteorological data: %s\n", filename);

  /* Open netCDF file... */
  if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {

    /* Try to stage meteo file... */
    if (ctl->met_stage[0] != '-') {
      sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
              year, mon, day, hour, filename);
      if (system(cmd) != 0)
        ERRMSG("Error while staging meteo data!");
    }

    /* Try to open again... */
    if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
      return 0;
  }

  /* Get time from filename... */
  sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
  hour = atoi(tstr);
  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx < 2 || nx > EX)
    ERRMSG("Number of longitudes out of range!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");

  sprintf(levname, "lev");
  NC(nc_inq_dimid(ncid, levname, &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
  if (np == 1) {
    sprintf(levname, "lev_2");
    NC(nc_inq_dimid(ncid, levname, &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
  }
  if (np < 2 || np > EP)
    ERRMSG("Number of levels out of range!");

  /* Store dimensions... */
  met->np = (int) np;
  met->nx = (int) nx;
  met->ny = (int) ny;

  /* Get horizontal grid... */
  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));
  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

  /* Read meteorological data... */
  read_met_help(ncid, "t", "T", met, met->t, 1.0);
  read_met_help(ncid, "u", "U", met, met->u, 1.0);
  read_met_help(ncid, "v", "V", met, met->v, 1.0);
  read_met_help(ncid, "w", "W", met, met->w, 0.01f);
  read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
  read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));

  /* Meteo data on pressure levels... */
  if (ctl->met_np <= 0) {

    /* Read pressure levels from file... */
    NC(nc_inq_varid(ncid, levname, &varid));
    NC(nc_get_var_double(ncid, varid, met->p));
    for (ip = 0; ip < met->np; ip++)
      met->p[ip] /= 100.;

    /* Extrapolate data for lower boundary... */
    read_met_extrapolate(met);
  }

  /* Meteo data on model levels... */
  else {

    /* Read pressure data from file... */
    read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);

    /* Interpolate from model levels to pressure levels... */
    read_met_ml2pl(ctl, met, met->t);
    read_met_ml2pl(ctl, met, met->u);
    read_met_ml2pl(ctl, met, met->v);
    read_met_ml2pl(ctl, met, met->w);
    read_met_ml2pl(ctl, met, met->h2o);
    read_met_ml2pl(ctl, met, met->o3);

    /* Set pressure levels... */
    met->np = ctl->met_np;
    for (ip = 0; ip < met->np; ip++)
      met->p[ip] = ctl->met_p[ip];
  }

  /* Check ordering of pressure levels... */
  for (ip = 1; ip < met->np; ip++)
    if (met->p[ip - 1] < met->p[ip])
      ERRMSG("Pressure levels must be descending!");

  /* Read surface pressure... */
  if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
      || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
  } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
             || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
  } else
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->ps[ix][iy] = met->p[0];

  /* Create periodic boundary conditions... */
  read_met_periodic(met);

  /* Calculate geopotential heights... */
  read_met_geopot(ctl, met);

  /* Calculate potential vorticity... */
  read_met_pv(met);

  /* Calculate tropopause pressure... */
  read_met_tropo(ctl, met);

  /* Downsampling... */
  read_met_sample(ctl, met);

  /* Close file... */
  NC(nc_close(ncid));

  /* Return success... */
  return 1;
}
```

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.

```
               {

  int ip, ip0, ix, iy;

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      for (ip0 = met->np - 1; ip0 >= 0; ip0--)
        if (!gsl_finite(met->t[ix][iy][ip0])
            || !gsl_finite(met->u[ix][iy][ip0])
            || !gsl_finite(met->v[ix][iy][ip0])
            || !gsl_finite(met->w[ix][iy][ip0]))
          break;

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--) {
        met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
        met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
        met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
        met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
        met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
        met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
      }
    }
}
```

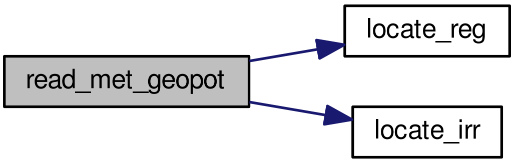
5.19.2.21 void read_met_geopot (ctl_t ∗ *ctl,* met_t ∗ *met*)

Calculate geopotential heights.

Definition at line 1551 of file libtrac.c.

```
               {

  static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];

  static int init, topo_nx = -1, topo_ny;

  FILE *in;

  char line[LEN];

  double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;

  float help[EX][EY];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;

  /* Initialize geopotential heights... */
#pragma omp parallel for default(shared) private(ix,iy,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++)
        met->z[ix][iy][ip] = GSL_NAN;

  /* Check filename... */
  if (ctl->met_geopot[0] == '-')
    return;

  /* Read surface geopotential... */
  if (!init) {
    init = 1;

    /* Write info... */
    printf("Read surface geopotential: %s\n", ctl->met_geopot);

    /* Open file... */
    if (!(in = fopen(ctl->met_geopot, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
        if (rlon != rlon_old) {
          if ((++topo_nx) >= EX)
            ERRMSG("Too many longitudes!");
          topo_ny = 0;
        }
        rlon_old = rlon;
        topo_lon[topo_nx] = rlon;
        topo_lat[topo_ny] = rlat;
        topo_z[topo_nx][topo_ny] = rz;
        if ((++topo_ny) >= EY)
          ERRMSG("Too many latitudes!");
      }
    if ((++topo_nx) >= EX)
      ERRMSG("Too many longitudes!");

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

    /* Check grid spacing... */
    if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
        || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
      printf("Warning: Grid spacing does not match!\n");
  }

  /* Apply hydrostatic equation to calculate geopotential heights... */
#pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Get surface height... */
      lon = met->lon[ix];
      if (lon < topo_lon[0])
        lon += 360;
      else if (lon > topo_lon[topo_nx - 1])
        lon -= 360;
      lat = met->lat[iy];
      tx = locate_reg(topo_lon, topo_nx, lon);
      ty = locate_reg(topo_lat, topo_ny, lat);
      z0 = LIN(topo_lon[tx], topo_z[tx][ty],
               topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
      z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
               topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
      z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);

      /* Find surface pressure level... */
      ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);

      /* Get surface temperature... */
      ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
               met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
                   * log(met->ps[ix][iy] / met->p[ip0 + 1]));
      for (ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                     * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
                     * log(met->p[ip - 1] / met->p[ip]));
    }

  /* Smooth fields... */
  for (ip = 0; ip < met->np; ip++) {

    /* Median filter... */
#pragma omp parallel for default(shared) private(ix,iy,n,ix2,ix3,iy2,data)
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {
        n = 0;
        for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
               iy2++)
            if (gsl_finite(met->z[ix3][iy2][ip])) {
              data[n] = met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0) {
          gsl_sort(data, 1, (size_t) n);
          help[ix][iy] = (float)
            gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
        } else
          help[ix][iy] = GSL_NAN;
      }

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

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.

```
             {

  float *help;

  int ip, ix, iy, varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      return;

  /* Allocate... */
  ALLOC(help, float, met->nx * met->ny * met->np);

  /* Read data... */
  NC(nc_get_var_float(ncid, varid, help));

  /* Copy and check data... */
#pragma omp parallel for default(shared) private(ix,iy,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
        if (fabsf(dest[ix][iy][ip]) < 1e14f)
          dest[ix][iy][ip] *= scl;
        else
          dest[ix][iy][ip] = GSL_NAN;
      }

  /* Free... */
  free(help);
}
```

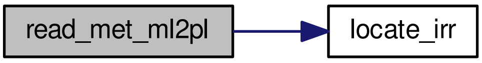
5.19.2.23 void read_met_ml2pl (ctl_t * *ctl,* met_t * *met,* float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 1736 of file libtrac.c.

```
                         {

  double aux[EP], p[EP], pt;

  int ip, ip2, ix, iy;

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Copy pressure profile... */
      for (ip = 0; ip < met->np; ip++)
        p[ip] = met->pl[ix][iy][ip];

      /* Interpolate... */
      for (ip = 0; ip < ctl->met_np; ip++) {
        pt = ctl->met_p[ip];
        if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
          pt = p[0];
        else if ((pt > p[met->np - 1] && p[1] > p[0])
                 || (pt < p[met->np - 1] && p[1] < p[0]))
          pt = p[met->np - 1];
        ip2 = locate_irr(p, met->np, pt);
        aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
                      p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
      }

      /* Copy data... */
      for (ip = 0; ip < ctl->met_np; ip++)
        var[ix][iy][ip] = (float) aux[ip];
    }
}
```

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.

```
               {

  int ip, iy;

  /* Check longitudes... */
  if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
             + met->lon[1] - met->lon[0] - 360) < 0.01))
    return;

  /* Increase longitude counter... */
  if ((++met->nx) > EX)
    ERRMSG("Cannot create periodic boundary conditions!");

  /* Set longitude... */
  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];

  /* Loop over latitudes and pressure levels... */
#pragma omp parallel for default(shared) private(iy,ip)
  for (iy = 0; iy < met->ny; iy++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->pt[met->nx - 1][iy] = met->pt[0][iy];
    for (ip = 0; ip < met->np; ip++) {
      met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
      met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
      met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
      met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
      met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
      met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
      met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
    }
  }
}
```

5.19.2.25 void read_met_pv (met_t * *met*)

Calculate potential vorticity.

Definition at line 1812 of file libtrac.c.

```
                {

  double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
    dtdp, dudp, dvdp, latr, vort, pows[EP];

  int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;

  /* Set powers... */
  for (ip = 0; ip < met->np; ip++)
    pows[ip] = pow(1000. / met->p[ip], 0.286);

  /* Loop over grid points... */
#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)
  for (ix = 0; ix < met->nx; ix++) {

    /* Set indices... */
    ix0 = GSL_MAX(ix - 1, 0);
    ix1 = GSL_MIN(ix + 1, met->nx - 1);

    /* Loop over grid points... */
    for (iy = 0; iy < met->ny; iy++) {

      /* Set indices... */
      iy0 = GSL_MAX(iy - 1, 0);
      iy1 = GSL_MIN(iy + 1, met->ny - 1);

      /* Set auxiliary variables... */
      latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
      dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
      dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
      c0 = cos(met->lat[iy0] / 180. * M_PI);
      c1 = cos(met->lat[iy1] / 180. * M_PI);
      cr = cos(latr / 180. * M_PI);
      vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);

      /* Loop over grid points... */
      for (ip = 0; ip < met->np; ip++) {

        /* Get gradients in longitude... */
        dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
        dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;

        /* Get gradients in latitude... */
        dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
        dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;

        /* Set indices... */
        ip0 = GSL_MAX(ip - 1, 0);
        ip1 = GSL_MIN(ip + 1, met->np - 1);

        /* Get gradients in pressure... */
        dp0 = 100. * (met->p[ip] - met->p[ip0]);
        dp1 = 100. * (met->p[ip1] - met->p[ip]);
        if (ip != ip0 && ip != ip1) {
          denom = dp0 * dp1 * (dp0 + dp1);
          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
                  - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
            / denom;
          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
                  - dp1 * dp1 * met->u[ix][iy][ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
            / denom;
          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
                  - dp1 * dp1 * met->v[ix][iy][ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
            / denom;
        } else {
          denom = dp0 + dp1;
          dtdp =
            (met->t[ix][iy][ip1] * pows[ip1] -
             met->t[ix][iy][ip0] * pows[ip0]) / denom;
          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
        }

        /* Calculate PV... */
        met->pv[ix][iy][ip] = (float)
          (1e6 * G0 *
           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
      }
    }
  }
}
```

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.

```
               {

  met_t *help;

  float w, wsum;

  int ip, ip2, ix, ix2, ix3, iy, iy2;

  /* Check parameters... */
  if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
      && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
    return;

  /* Allocate... */
  ALLOC(help, met_t, 1);

  /* Copy data... */
  help->nx = met->nx;
  help->ny = met->ny;
  help->np = met->np;
  memcpy(help->lon, met->lon, sizeof(met->lon));
  memcpy(help->lat, met->lat, sizeof(met->lat));
  memcpy(help->p, met->p, sizeof(met->p));

  /* Smoothing... */
  for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
    for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
      for (ip = 0; ip < met->np; ip += ctl->met_dp) {
        help->ps[ix][iy] = 0;
        help->pt[ix][iy] = 0;
        help->z[ix][iy][ip] = 0;
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
        help->pv[ix][iy][ip] = 0;
        help->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        wsum = 0;
        for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
            for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
              w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
                * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
                * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->pt[ix][iy] += w * met->pt[ix3][iy2];
              help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
              help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
              help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
              help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
              help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
              help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
              help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->pt[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->z[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->pv[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Downsampling... */
  met->nx = 0;
  for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
      met->lat[met->ny] = help->lat[iy];
      met->ps[met->nx][met->ny] = help->ps[ix][iy];
      met->pt[met->nx][met->ny] = help->pt[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl->met_dp) {
        met->p[met->np] = help->p[ip];
        met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
        met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
        met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
        met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
        met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
        met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
        met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
        met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

  /* Free... */
  free(help);
}
```

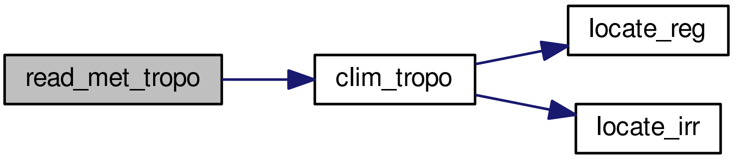
5.19.2.27 void read_met_tropo (ctl_t * *ctl,* met_t * *met*)

Calculate tropopause pressure.

Definition at line 2015 of file libtrac.c.

```
             {

gsl_interp_accel *acc;

gsl_spline *spline;

double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
  z[400], z2[400];

int found, ix, iy, iz, iz2;

/* Allocate... */
acc = gsl_interp_accel_alloc();
spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);

/* Get altitude and pressure profiles... */
for (iz = 0; iz < met->np; iz++)
  z[iz] = Z(met->p[iz]);
for (iz = 0; iz <= 170; iz++) {
  z2[iz] = 4.5 + 0.1 * iz;
  p2[iz] = P(z2[iz]);
}

/* Do not calculate tropopause... */
if (ctl->met_tropo == 0)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      met->pt[ix][iy] = GSL_NAN;

/* Use tropopause climatology... */
else if (ctl->met_tropo == 1)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);

/* Use cold point... */
else if (ctl->met_tropo == 2) {

  /* Loop over grid points... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Interpolate temperature profile... */
      for (iz = 0; iz < met->np; iz++)
        t[iz] = met->t[ix][iy][iz];
      gsl_spline_init(spline, z, t, (size_t) met->np);
      for (iz = 0; iz <= 170; iz++)
        t2[iz] = gsl_spline_eval(spline, z2[iz], acc);

      /* Find minimum... */
      iz = (int) gsl_stats_min_index(t2, 1, 171);
      if (iz <= 0 || iz >= 170)
        met->pt[ix][iy] = GSL_NAN;
      else
        met->pt[ix][iy] = p2[iz];
    }
}

/* Use WMO definition... */
else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {

  /* Loop over grid points... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Interpolate temperature profile... */
      for (iz = 0; iz < met->np; iz++)
        t[iz] = met->t[ix][iy][iz];
      gsl_spline_init(spline, z, t, (size_t) met->np);
      for (iz = 0; iz <= 160; iz++)
        t2[iz] = gsl_spline_eval(spline, z2[iz], acc);

      /* Find 1st tropopause... */
      met->pt[ix][iy] = GSL_NAN;
      for (iz = 0; iz <= 140; iz++) {
        found = 1;
        for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
          if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
              / log(p2[iz2] / p2[iz]) > 2.0) {
            found = 0;
            break;
          }
        if (found) {
          if (iz > 0 && iz < 140)
            met->pt[ix][iy] = p2[iz];
          break;
        }
        }

        /* Find 2nd tropopause... */
        if (ctl->met_tropo == 4) {
          met->pt[ix][iy] = GSL_NAN;
          for (; iz <= 140; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
              if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                  / log(p2[iz2] / p2[iz]) < 3.0) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 140; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
              if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                  / log(p2[iz2] / p2[iz]) > 2.0) {
                found = 0;
                break;
              }
            if (found) {
              if (iz > 0 && iz < 140)
                met->pt[ix][iy] = p2[iz];
              break;
            }
          }
        }
      }
  }

  /* Use dynamical tropopause... */
  else if (ctl->met_tropo == 5) {

    /* Loop over grid points... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {

        /* Interpolate potential vorticity profile... */
        for (iz = 0; iz < met->np; iz++)
          pv[iz] = met->pv[ix][iy][iz];
        gsl_spline_init(spline, z, pv, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Interpolate potential temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
        gsl_spline_init(spline, z, th, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          th2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Find dynamical tropopause 3.5 PVU + 380 K */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 160; iz++)
          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
            if (iz > 0 && iz < 160)
              met->pt[ix][iy] = p2[iz];
            break;
          }
      }
  }

  else
    ERRMSG("Cannot calculate tropopause!");

  /* Free... */
  gsl_spline_free(spline);
  gsl_interp_accel_free(acc);
}
```

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.

```
               {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    msg[2 * LEN], rvarname[LEN], rval[LEN];

  int contain = 0, i;

  /* Open file... */
  if (filename[strlen(filename) - 1] != '-')
    if (!(in = fopen(filename, "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
        }
  for (i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else {
      sprintf(msg, "Missing variable %s!\n", fullname1);
      ERRMSG(msg);
    }
  }

  /* Write info... */
  printf("%s = %s\n", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}
```

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.

```
                {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

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.

```
            {

  static double starttime[NTIMER], runtime[NTIMER];

  /* Check id... */
  if (id < 0 || id >= NTIMER)
    ERRMSG("Too many timers!");

  /* Start timer... */
  if (mode == 1) {
    if (starttime[id] <= 0)
      starttime[id] = omp_get_wtime();
    else
      ERRMSG("Timer already started!");
  }

  /* Stop timer... */
  else if (mode == 2) {
    if (starttime[id] > 0) {
      runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
      starttime[id] = -1;
    }
  }

  /* Print timer... */
  else if (mode == 3) {
    printf("%s = %.3f s\n", name, runtime[id]);
    runtime[id] = 0;
  }
}
```

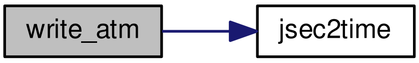
5.19.2.31 void write_atm (const char ∗ *filename,* ctl_t ∗ *ctl,* atm_t ∗ *atm,* double *t*)

Write atmospheric data.

Definition at line 2319 of file libtrac.c.

```
            {

  FILE *in, *out;

  char line[LEN];

  double r, t0, t1;

  int ip, iq, year, mon, day, hour, min, sec;

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Write info... */
  printf("Write atmospheric data: %s\n", filename);

  /* Write ASCII data... */
  if (ctl->atm_type == 0) {

    /* Check if gnuplot output is requested... */
    if (ctl->atm_gpfile[0] != '-') {

      /* Create gnuplot pipe... */
      if (!(out = popen("gnuplot", "w")))
        ERRMSG("Cannot create pipe to gnuplot!");

      /* Set plot filename... */
      fprintf(out, "set out \"%s.png\"\n", filename);

      /* Set time string... */
      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
      fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
              year, mon, day, hour, min);

      /* Dump gnuplot file to pipe... */
      if (!(in = fopen(ctl->atm_gpfile, "r")))
        ERRMSG("Cannot open file!");
      while (fgets(line, LEN, in))
        fprintf(out, "%s", line);
      fclose(in);
    }

    else {

      /* Create file... */
      if (!(out = fopen(filename, "w")))
        ERRMSG("Cannot create file!");
    }

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
              ctl->qnt_unit[iq]);
    fprintf(out, "\n");

    /* Write data... */
    for (ip = 0; ip < atm->np; ip++) {

      /* Check time... */
      if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
        continue;

      /* Write output... */
      fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
              atm->lon[ip], atm->lat[ip]);
      for (iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
      }
      fprintf(out, "\n");
    }

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

  /* Write binary data... */
  else if (ctl->atm_type == 1) {

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write data... */
    FWRITE(&atm->np, int,
           1,
           out);
    FWRITE(atm->time, double,
              (size_t) atm->np,
           out);
    FWRITE(atm->p, double,
              (size_t) atm->np,
           out);
    FWRITE(atm->lon, double,
              (size_t) atm->np,
           out);
    FWRITE(atm->lat, double,
              (size_t) atm->np,
           out);
    for (iq = 0; iq < ctl->nq; iq++)
      FWRITE(atm->q[iq], double,
                (size_t) atm->np,
             out);

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

  /* Error... */
  else
    ERRMSG("Atmospheric data type not supported!");
}
```

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.

```
              {

  static FILE *in, *out;

  static char line[LEN];

  static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
    rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;

  static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantity index for mass... */
    if (ctl->qnt_m < 0)
      ERRMSG("Need quantity mass!");

    /* Open observation data file... */
    printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
    if (!(in = fopen(ctl->csi_obsfile, "r")))
      ERRMSG("Cannot open file!");

    /* Create new file... */
    printf("Write CSI data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = number of hits (cx)\n"
            "# $3 = number of misses (cy)\n"
            "# $4 = number of false alarms (cz)\n"
            "# $5 = number of observations (cx + cy)\n"
            "# $6 = number of forecasts (cx + cz)\n"
            "# $7 = bias (forecasts/observations) [%%]\n"
            "# $8 = probability of detection (POD) [%%]\n"
            "# $9 = false alarm rate (FAR) [%%]\n"
            "# $10 = critical success index (CSI) [%%]\n\n");
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Initialize grid cells... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->csi_nx; ix++)
    for (iy = 0; iy < ctl->csi_ny; iy++)
      for (iz = 0; iz < ctl->csi_nz; iz++)
        modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;

  /* Read observation data... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
        5)
      continue;

    /* Check time... */
    if (rt < t0)
      continue;
    if (rt > t1)
      break;

    /* Calculate indices... */
    ix = (int) ((rlon - ctl->csi_lon0)
                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
    iy = (int) ((rlat - ctl->csi_lat0)
                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
    iz = (int) ((rz - ctl->csi_z0)
                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->csi_nx ||
        iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
      continue;

    /* Get mean observation index... */
    obsmean[ix][iy][iz] += robs;
    obscount[ix][iy][iz]++;
  }

  /* Analyze model data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Get indices... */
    ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
    iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
    iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->csi_nx ||
        iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
      continue;

    /* Get total mass in grid cell... */
    modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  }

  /* Analyze all grid cells... */
#pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
  for (ix = 0; ix < ctl->csi_nx; ix++)
    for (iy = 0; iy < ctl->csi_ny; iy++)
      for (iz = 0; iz < ctl->csi_nz; iz++) {

        /* Calculate mean observation index... */
        if (obscount[ix][iy][iz] > 0)
          obsmean[ix][iy][iz] /= obscount[ix][iy][iz];

        /* Calculate column density... */
        if (modmean[ix][iy][iz] > 0) {
          dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
          dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
          lat = ctl->csi_lat0 + dlat * (iy + 0.5);
          area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
            * cos(lat * M_PI / 180.);
          modmean[ix][iy][iz] /= (1e6 * area);
        }

        /* Calculate CSI... */
        if (obscount[ix][iy][iz] > 0) {
          if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
              modmean[ix][iy][iz] >= ctl->csi_modmin)
            cx++;
          else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                   modmean[ix][iy][iz] < ctl->csi_modmin)
            cy++;
          else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                   modmean[ix][iy][iz] >= ctl->csi_modmin)
            cz++;
        }
      }

  /* Write output... */
  if (fmod(t, ctl->csi_dt_out) == 0) {

    /* Write... */
    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
            t, cx, cy, cz, cx + cy, cx + cz,
            (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
            (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
            (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
            (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);

    /* Set counters to zero... */
    cx = cy = cz = 0;
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

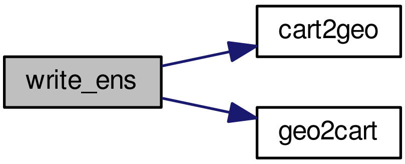
5.19.2.33 void write_ens (const char ∗ *filename*, ctl_t ∗ *ctl*, atm_t ∗ *atm*, double *t*)

Write ensemble data.

Definition at line 2613 of file libtrac.c.

```
            {

  static FILE *out;

  static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
    t0, t1, x[NENS][3], xm[3];

  static int ip, iq;

  static size_t i, n;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantities... */
    if (ctl->qnt_ens < 0)
      ERRMSG("Missing ensemble IDs!");

    /* Create new file... */
    printf("Write ensemble data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Init... */
  ens = GSL_NAN;
  n = 0;

  /* Loop over air parcels... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Check ensemble id... */
    if (atm->q[ctl->qnt_ens][ip] != ens) {

      /* Write results... */
      if (n > 0) {

        /* Get mean position... */
        xm[0] = xm[1] = xm[2] = 0;
        for (i = 0; i < n; i++) {
          xm[0] += x[i][0] / (double) n;
          xm[1] += x[i][1] / (double) n;
          xm[2] += x[i][2] / (double) n;
        }
        cart2geo(xm, &dummy, &lon, &lat);
        fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
                lat);

        /* Get quantity statistics... */
        for (iq = 0; iq < ctl->nq; iq++) {
          fprintf(out, " ");
          fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
        }
        for (iq = 0; iq < ctl->nq; iq++) {
          fprintf(out, " ");
          fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
        }
        fprintf(out, " %lu\n", n);
      }

      /* Init new ensemble... */
      ens = atm->q[ctl->qnt_ens][ip];
      n = 0;
    }

    /* Save data... */
    p[n] = atm->p[ip];
    geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
    for (iq = 0; iq < ctl->nq; iq++)
      q[iq][n] = atm->q[iq][ip];
    if ((++n) >= NENS)
      ERRMSG("Too many data points!");
  }

  /* Write results... */
  if (n > 0) {

    /* Get mean position... */
    xm[0] = xm[1] = xm[2] = 0;
    for (i = 0; i < n; i++) {
      xm[0] += x[i][0] / (double) n;
      xm[1] += x[i][1] / (double) n;
      xm[2] += x[i][2] / (double) n;
    }
    cart2geo(xm, &dummy, &lon, &lat);
    fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);

    /* Get quantity statistics... */
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
    }
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
    }
    fprintf(out, " %lu\n", n);
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

Here is the call graph for this function:



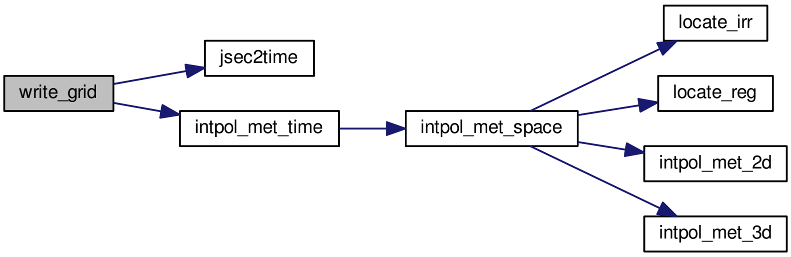
5.19.2.34 void write_grid (const char ∗ *filename,* ctl_t ∗ *ctl,* met_t ∗ *met0,* met_t ∗ *met1,* atm_t ∗ *atm,* double *t*)

Write gridded data.

Definition at line 2744 of file libtrac.c.

```
      {

  FILE *in, *out;

  char line[LEN];

  static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
    area, rho_air, press, temp, cd, vmr, t0, t1, r;

  static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;

  /* Check dimensions... */
  if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Set grid box size... */
  dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
  dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
  dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;

  /* Initialize grid... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->grid_nx; ix++)
    for (iy = 0; iy < ctl->grid_ny; iy++)
      for (iz = 0; iz < ctl->grid_nz; iz++) {
        mass[ix][iy][iz] = 0;
        np[ix][iy][iz] = 0;
      }

  /* Average data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++)
    if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {

      /* Get index... */
      ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
      iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
      iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);

      /* Check indices... */
      if (ix < 0 || ix >= ctl->grid_nx ||
          iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
        continue;

      /* Add mass... */
      if (ctl->qnt_m >= 0)
        mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
      np[ix][iy][iz]++;
    }

  /* Check if gnuplot output is requested... */
  if (ctl->grid_gpfile[0] != '-') {

    /* Write info... */
    printf("Plot grid data: %s.png\n", filename);

    /* Create gnuplot pipe... */
    if (!(out = popen("gnuplot", "w")))
      ERRMSG("Cannot create pipe to gnuplot!");

    /* Set plot filename... */
    fprintf(out, "set out \"%s.png\"\n", filename);

    /* Set time string... */
    jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
    fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
            year, mon, day, hour, min);

    /* Dump gnuplot file to pipe... */
    if (!(in = fopen(ctl->grid_gpfile, "r")))
      ERRMSG("Cannot open file!");
    while (fgets(line, LEN, in))
      fprintf(out, "%s", line);
    fclose(in);
  }

  else {

    /* Write info... */
    printf("Write grid data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");
  }

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = surface area [km^2]\n"
          "# $6 = layer width [km]\n"
          "# $7 = number of particles [1]\n"
          "# $8 = column density [kg/m^2]\n"
          "# $9 = volume mixing ratio [1]\n\n");

  /* Write data... */
  for (ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
      fprintf(out, "\n");
    for (iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
        fprintf(out, "\n");
      for (iz = 0; iz < ctl->grid_nz; iz++)
        if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {

          /* Set coordinates... */
          z = ctl->grid_z0 + dz * (iz + 0.5);
          lon = ctl->grid_lon0 + dlon * (ix + 0.5);
          lat = ctl->grid_lat0 + dlat * (iy + 0.5);

          /* Get pressure and temperature... */
          press = P(z);
          intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                          NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);

          /* Calculate surface area... */
          area = dlat * dlon * SQR(RE * M_PI / 180.)
            * cos(lat * M_PI / 180.);

          /* Calculate column density... */
          cd = mass[ix][iy][iz] / (1e6 * area);

          /* Calculate volume mixing ratio... */
          rho_air = 100. * press / (RA * temp);
          vmr = MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * 1e6 * area * 1e3 * dz);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
                  t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
        }
    }
  }

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

Here is the call graph for this function:



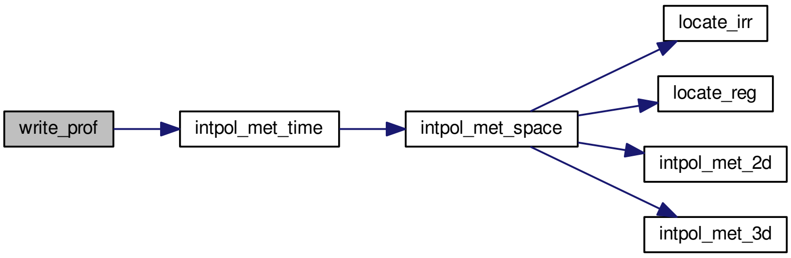
5.19.2.35 void write_prof (const char * *filename*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

Write profile data.

Definition at line 2897 of file libtrac.c.

```
            {

  static FILE *in, *out;

  static char line[LEN];

  static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
    rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
    rho_air, vmr, h2o, o3;

  static int obscount[GX][GY], ip, ix, iy, iz, okay;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantity index for mass... */
    if (ctl->qnt_m < 0)
      ERRMSG("Need quantity mass!");

    /* Check dimensions... */
    if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
      ERRMSG("Grid dimensions too large!");

    /* Open observation data file... */
    printf("Read profile observation data: %s\n", ctl->prof_obsfile);
    if (!(in = fopen(ctl->prof_obsfile, "r")))
      ERRMSG("Cannot open file!");

    /* Create new output file... */
    printf("Write profile data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n"
            "# $4 = latitude [deg]\n"
            "# $5 = pressure [hPa]\n"
            "# $6 = temperature [K]\n"
            "# $7 = volume mixing ratio [1]\n"
            "# $8 = H2O volume mixing ratio [1]\n"
            "# $9 = O3 volume mixing ratio [1]\n"
            "# $10 = observed BT index (mean) [K]\n"
            "# $11 = observed BT index (sigma) [K]\n");

    /* Set grid box size... */
    dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
    dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
    dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Initialize... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++) {
      obsmean[ix][iy] = 0;
      obsmean2[ix][iy] = 0;
      obscount[ix][iy] = 0;
      for (iz = 0; iz < ctl->prof_nz; iz++)
        mass[ix][iy][iz] = 0;
    }

  /* Read observation data... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
        5)
      continue;

    /* Check time... */
    if (rt < t0)
      continue;
    if (rt > t1)
      break;

    /* Calculate indices... */
    ix = (int) ((rlon - ctl->prof_lon0) / dlon);
    iy = (int) ((rlat - ctl->prof_lat0) / dlat);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
      continue;

    /* Get mean observation index... */
    obsmean[ix][iy] += robs;
    obsmean2[ix][iy] += SQR(robs);
    obscount[ix][iy]++;
  }

  /* Analyze model data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Get indices... */
    ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
    iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
    iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->prof_nx ||
        iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
      continue;

    /* Get total mass in grid cell... */
    mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  }

  /* Extract profiles... */
  for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++)
      if (obscount[ix][iy] > 0) {

        /* Check profile... */
        okay = 0;
        for (iz = 0; iz < ctl->prof_nz; iz++)
          if (mass[ix][iy][iz] > 0) {
            okay = 1;
            break;
          }
        if (!okay)
          continue;

        /* Write output... */
        fprintf(out, "\n");

        /* Loop over altitudes... */
        for (iz = 0; iz < ctl->prof_nz; iz++) {

          /* Set coordinates... */
          z = ctl->prof_z0 + dz * (iz + 0.5);
          lon = ctl->prof_lon0 + dlon * (ix + 0.5);
          lat = ctl->prof_lat0 + dlat * (iy + 0.5);

          /* Get pressure and temperature... */
          press = P(z);
          intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                          NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);

          /* Calculate surface area... */
          area = dlat * dlon * SQR(M_PI * RE / 180.)
            * cos(lat * M_PI / 180.);

          /* Calculate volume mixing ratio... */
          rho_air = 100. * press / (RA * temp);
          vmr = MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * area * dz * 1e9);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
                  t, z, lon, lat, press, temp, vmr, h2o, o3,
                  obsmean[ix][iy] / obscount[ix][iy],
                  sqrt(obsmean2[ix][iy] / obscount[ix][iy]
                       - SQR(obsmean[ix][iy] / obscount[ix][iy])));
        }
      }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

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*)

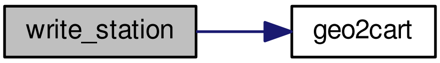
Write station data.

Definition at line 3077 of file libtrac.c.

```
          {

static FILE *out;

static double rmax2, t0, t1, x0[3], x1[3];

static int ip, iq;

/* Init... */
if (t == ctl->t_start) {

  /* Write info... */
  printf("Write station data: %s\n", filename);

  /* Create new file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
  for (iq = 0; iq < ctl->nq; iq++)
    fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
            ctl->qnt_name[iq], ctl->qnt_unit[iq]);
  fprintf(out, "\n");

  /* Set geolocation and search radius... */
  geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
  rmax2 = SQR(ctl->stat_r);
}

/* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;

/* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {

  /* Check time... */
  if (atm->time[ip] < t0 || atm->time[ip] > t1)
    continue;

  /* Check station flag... */
  if (ctl->qnt_stat >= 0)
    if (atm->q[ctl->qnt_stat][ip])
      continue;

  /* Get Cartesian coordinates... */
```

```
    geo2cart(0, atm->lon[ip], atm->lat[ip], x1);

    /* Check horizontal distance... */
    if (DIST2(x0, x1) > rmax2)
      continue;

    /* Set station flag... */
    if (ctl->qnt_stat >= 0)
      atm->q[ctl->qnt_stat][ip] = 1;

    /* Write data... */
    fprintf(out, "%.2f %g %g %g",
            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

Here is the call graph for this function:



5.20 libtrac.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

/*****************************************************************************/

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

  double radius;

  radius = NORM(x);
  *lat = asin(x[2] / radius) * 180 / M_PI;
  *lon = atan2(x[1], x[0]) * 180 / M_PI;
  *z = radius - RE;
}
```

```

/*****************************************************************************/

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

  static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
    9072000.00, 11664000.00, 14342400.00,
    16934400.00, 19612800.00, 22291200.00,
    24883200.00, 27561600.00, 30153600.00
  };

  static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
    5, 15, 25, 35, 45, 55, 65, 75, 85
  };

  static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
    31.6228, 46.4159, 68.1292, 100, 146.78
  };

  static double hno3[12][18][10] = {
    {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
     {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
     {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
     {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
     {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
     {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
     {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
     {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
     {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
     {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
     {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
     {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
     {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
     {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
     {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
     {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
     {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
     {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
    {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
     {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
     {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
     {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
     {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
     {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
     {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
     {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
     {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
     {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
     {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
     {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
     {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
     {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
     {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
     {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
     {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
     {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
    {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
     {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
     {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
     {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
     {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
     {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
     {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
     {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
     {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
     {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
     {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
     {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
     {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
     {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
     {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
     {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
     {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
     {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
    {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
     {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
     {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
     {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
     {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
     {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
     {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
     {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
     {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
     {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
     {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
     {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
     {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
     {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
     {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
     {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
     {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
     {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
    {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
     {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
     {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
     {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
     {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
     {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
     {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
     {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
     {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
     {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
     {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
     {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
     {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
     {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
     {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
     {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
     {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
     {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
    {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
     {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
     {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
     {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
     {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
     {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
     {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
     {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
     {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
     {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
     {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
     {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
     {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
     {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
     {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
     {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
     {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
     {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
    {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
     {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
     {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
     {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
     {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
     {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
     {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
     {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
     {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
     {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
     {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
     {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
     {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
     {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
     {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
     {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
     {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
     {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
    {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
     {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
     {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
     {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
     {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
     {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
     {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
     {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
     {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
     {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
     {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
    {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
    {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
    {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
    {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
  };

  double aux00, aux01, aux10, aux11, sec;

  int ilat, ip, isec;

  /* Get seconds since begin of year... */
  sec = fmod(t, 365.25 * 86400.);

  /* Get indices... */
  isec = locate_irr(secs, 12, sec);
  ilat = locate_reg(lats, 18, lat);
  ip = locate_irr(ps, 10, p);

  /* Interpolate... */
  aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
              ps[ip + 1], hno3[isec][ilat][ip + 1], p);
  aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
              ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
  aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
              ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
  aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
              ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
  aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
  return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
}

/*****************************************************************************/

double clim_tropo(
  double t,
  double lat) {

  static double doys[12]
  = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };

  static double lats[73]
    = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
    -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
    -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
    -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
    15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
    45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
    75, 77.5, 80, 82.5, 85, 87.5, 90
  };

  static double tps[12][73]
    = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
         297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
         175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
         99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
         98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
         152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
         277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
         275.3, 275.6, 275.4, 274.1, 273.5},
  {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
   300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
   150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
   98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
   98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
   220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
   284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
   287.5, 286.2, 285.8},
  {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
   297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
   161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
   100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
   99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
   186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
   279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
   304.3, 304.9, 306, 306.6, 306.2, 306},
  {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
   290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
   195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
   102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
   99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
   148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
   263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
   315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
  {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
   260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
   205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
   101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
   102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
   165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
   273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
   325.3, 325.8, 325.8},
  {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
   222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
   228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
   105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
   106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
   127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
   251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
   308.5, 312.2, 313.1, 313.3},
  {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
   187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
   235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
   110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
   111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
   117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
   224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
   275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
  {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
   185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
   233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
   110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
   112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
   120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
   230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
   278.2, 282.6, 287.4, 290.9, 292.5, 293},
  {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
   183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
   243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
   114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
   110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
   114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
   203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
   276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
  {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
   215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
   237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
   111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
   106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
   112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
   206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
   279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
   305.1},
  {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
   253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
   223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
   108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
   102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
   109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
   241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
   286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
  {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
   284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
   175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
   100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
   100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
   186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
   280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
   281.7, 281.1, 281.2}
  };

  double doy, p0, p1;

  int imon, ilat;

  /* Get day of year... */
  doy = fmod(t / 86400., 365.25);
  while (doy < 0)
    doy += 365.25;

  /* Get indices... */
  ilat = locate_reg(lats, 73, lat);
  imon = locate_irr(doys, 12, doy);

  /* Interpolate... */
  p0 = LIN(lats[ilat], tps[imon][ilat],
           lats[ilat + 1], tps[imon][ilat + 1], lat);
  p1 = LIN(lats[ilat], tps[imon + 1][ilat],
           lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
  return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
}

/*****************************************************************************/

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

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

  /* Get day of year... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
    *doy = d0l[mon - 1] + day - 1;
  else
    *doy = d0[mon - 1] + day - 1;
}

/*****************************************************************************/

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

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
  int i;

  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d01[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d01[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}

/*****************************************************************************/

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

  double radius;

  radius = z + RE;
  x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
  x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
  x[2] = radius * sin(lat / 180 * M_PI);
}

/*****************************************************************************/

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

  static int init, ip, ix, iy;

  met_t *mets;

  char filename[LEN];

  /* Init... */
  if (t == ctl->t_start || !init) {
    init = 1;

    get_met_help(t, -1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met0))
      ERRMSG("Cannot open file!");

    get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met1))
      ERRMSG("Cannot open file!");
  }

  /* Read new data for forward trajectories... */
  if (t > (*met1)->time && ctl->direction == 1) {
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;
    get_met_help(t, 1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met1))
      ERRMSG("Cannot open file!");
  }

  /* Read new data for backward trajectories... */
  if (t < (*met0)->time && ctl->direction == -1) {
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;
    get_met_help(t, -1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met0))
      ERRMSG("Cannot open file!");
  }

  /* Check that grids are consistent... */
  if ((*met0)->nx != (*met1)->nx
      || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
    ERRMSG("Meteo grid dimensions do not match!");
  for (ix = 0; ix < (*met0)->nx; ix++)
    if ((*met0)->lon[ix] != (*met1)->lon[ix])
      ERRMSG("Meteo grid longitudes do not match!");
  for (iy = 0; iy < (*met0)->ny; iy++)
    if ((*met0)->lat[iy] != (*met1)->lat[iy])
      ERRMSG("Meteo grid latitudes do not match!");
  for (ip = 0; ip < (*met0)->np; ip++)
    if ((*met0)->p[ip] != (*met1)->p[ip])
      ERRMSG("Meteo grid pressure levels do not match!");
}

/*****************************************************************************/

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

  char repl[LEN];

  double t6, r;

  int year, mon, day, hour, min, sec;

  /* Round time to fixed intervals... */
  if (direct == -1)
    t6 = floor(t / dt_met) * dt_met;
  else
    t6 = ceil(t / dt_met) * dt_met;

  /* Decode time... */
  jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Set filename... */
  sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
  sprintf(repl, "%d", year);
  get_met_replace(filename, "YYYY", repl);
  sprintf(repl, "%02d", mon);
  get_met_replace(filename, "MM", repl);
  sprintf(repl, "%02d", day);
  get_met_replace(filename, "DD", repl);
  sprintf(repl, "%02d", hour);
  get_met_replace(filename, "HH", repl);
}

/*****************************************************************************/

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

  char buffer[LEN], *ch;

  int i;

  /* Iterate... */
  for (i = 0; i < 3; i++) {

    /* Replace substring... */
    if (!(ch = strstr(orig, search)))
      return;
    strncpy(buffer, orig, (size_t) (ch - orig));
    buffer[ch - orig] = 0;
    sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
    orig[0] = 0;
    strcpy(orig, buffer);
  }
}

/*****************************************************************************/

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

  double aux00, aux01, aux10, aux11;

  /* Set variables... */
  aux00 = array[ix][iy];
  aux01 = array[ix][iy + 1];
  aux10 = array[ix + 1][iy];
  aux11 = array[ix + 1][iy + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}

/*****************************************************************************/

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

  double aux00, aux01, aux10, aux11;

  /* Interpolate vertically... */
  aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
    + array[ix][iy][ip + 1];
  aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
    + array[ix][iy + 1][ip + 1];
  aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
    + array[ix + 1][iy][ip + 1];
  aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
    + array[ix + 1][iy + 1][ip + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}

/*****************************************************************************/

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) {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (met->lon[met->nx - 1] > 180 && lon < 0)
    lon += 360;

  /* Get indices... */
  ip = locate_irr(met->p, met->np, p);
  ix = locate_reg(met->lon, met->nx, lon);
  iy = locate_reg(met->lat, met->ny, lat);

  /* Get weights... */
  wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
  wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
  wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);

  /* Interpolate... */
  if (ps != NULL)
    intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
  if (pt != NULL)
    intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
  if (z != NULL)
    intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
  if (t != NULL)
    intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
  if (u != NULL)
    intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
  if (v != NULL)
```

```
    intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
  if (w != NULL)
    intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
  if (pv != NULL)
    intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
  if (h2o != NULL)
    intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
  if (o3 != NULL)
    intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
}

/*****************************************************************************/

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) {

  double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
    v0, v1, w0, w1, wt, z0, z1;

  /* Spatial interpolation... */
  intpol_met_space(met0, p, lon, lat,
                   ps == NULL ? NULL : &ps0,
                   pt == NULL ? NULL : &pt0,
                   z == NULL ? NULL : &z0,
                   t == NULL ? NULL : &t0,
                   u == NULL ? NULL : &u0,
                   v == NULL ? NULL : &v0,
                   w == NULL ? NULL : &w0,
                   pv == NULL ? NULL : &pv0,
                   h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
  intpol_met_space(met1, p, lon, lat,
                   ps == NULL ? NULL : &ps1,
                   pt == NULL ? NULL : &pt1,
                   z == NULL ? NULL : &z1,
                   t == NULL ? NULL : &t1,
                   u == NULL ? NULL : &u1,
                   v == NULL ? NULL : &v1,
                   w == NULL ? NULL : &w1,
                   pv == NULL ? NULL : &pv1,
                   h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);

  /* Get weighting factor... */
  wt = (met1->time - ts) / (met1->time - met0->time);

  /* Interpolate... */
  if (ps != NULL)
    *ps = wt * (ps0 - ps1) + ps1;
  if (pt != NULL)
    *pt = wt * (pt0 - pt1) + pt1;
  if (z != NULL)
    *z = wt * (z0 - z1) + z1;
  if (t != NULL)
    *t = wt * (t0 - t1) + t1;
  if (u != NULL)
    *u = wt * (u0 - u1) + u1;
  if (v != NULL)
    *v = wt * (v0 - v1) + v1;
  if (w != NULL)
    *w = wt * (w0 - w1) + w1;
  if (pv != NULL)
    *pv = wt * (pv0 - pv1) + pv1;
  if (h2o != NULL)
    *h2o = wt * (h2o0 - h2o1) + h2o1;
  if (o3 != NULL)
    *o3 = wt * (o30 - o31) + o31;
}

/*****************************************************************************/

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

  struct tm t0, *t1;

  time_t jsec0;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}

/*****************************************************************************/

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

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}

/*****************************************************************************/

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

  int i;

  /* Calculate index... */
  i = (int) ((x - xx[0]) / (xx[1] - xx[0]));

  /* Check range... */
  if (i < 0)
    i = 0;
  else if (i >= n - 2)
    i = n - 2;

  return i;
}

/*****************************************************************************/

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

  FILE *in;

  char line[LEN], *tok;

  double t0;

  int dimid, ip, iq, ncid, varid;

  size_t nparts;

   /* Init... */
  atm->np = 0;

   /* Write info... */
  printf("Read atmospheric data: %s\n", filename);

   /* Read ASCII data... */
  if (ctl->atm_type == 0) {

    /* Open file... */
    if (!(in = fopen(filename, "r")))
      return 0;

     /* Read line... */
    while (fgets(line, LEN, in)) {

      /* Read data... */
      TOK(line, tok, "%lg", atm->time[atm->np]);
      TOK(NULL, tok, "%lg", atm->p[atm->np]);
      TOK(NULL, tok, "%lg", atm->lon[atm->np]);
      TOK(NULL, tok, "%lg", atm->lat[atm->np]);
      for (iq = 0; iq < ctl->nq; iq++)
        TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

       /* Convert altitude to pressure... */
       atm->p[atm->np] = P(atm->p[atm->np]);

       /* Increment data point counter... */
      if ((++atm->np) > NP)
        ERRMSG("Too many data points!");
    }

    /* Close file... */
    fclose(in);
  }

   /* Read binary data... */
   else if (ctl->atm_type == 1) {

    /* Open file... */
    if (!(in = fopen(filename, "r")))
      return 0;

    /* Read data... */
    FREAD(&atm->np, int, 1, in);
    FREAD(atm->time, double,
            (size_t) atm->np,
          in);
    FREAD(atm->p, double,
            (size_t) atm->np,
          in);
    FREAD(atm->lon, double,
            (size_t) atm->np,
          in);
    FREAD(atm->lat, double,
            (size_t) atm->np,
          in);
    for (iq = 0; iq < ctl->nq; iq++)
      FREAD(atm->q[iq], double,
              (size_t) atm->np,
            in);

    /* Close file... */
    fclose(in);
  }

   /* Read netCDF data... */
   else if (ctl->atm_type == 2) {

    /* Open file... */
    if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
      return 0;

     /* Get dimensions... */
```

```
    NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nparts));
    atm->np = (int) nparts;
    if (atm->np > NP)
      ERRMSG("Too many particles!");

    /* Get time... */
    NC(nc_inq_varid(ncid, "time", &varid));
    NC(nc_get_var_double(ncid, varid, &t0));
    for (ip = 0; ip < atm->np; ip++)
      atm->time[ip] = t0;

    /* Read geolocations... */
    NC(nc_inq_varid(ncid, "PRESS", &varid));
    NC(nc_get_var_double(ncid, varid, atm->p));
    NC(nc_inq_varid(ncid, "LON", &varid));
    NC(nc_get_var_double(ncid, varid, atm->lon));
    NC(nc_inq_varid(ncid, "LAT", &varid));
    NC(nc_get_var_double(ncid, varid, atm->lat));

    /* Read variables... */
    if (ctl->qnt_p >= 0)
      if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
    if (ctl->qnt_t >= 0)
      if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
    if (ctl->qnt_u >= 0)
      if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
    if (ctl->qnt_v >= 0)
      if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
    if (ctl->qnt_w >= 0)
      if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
    if (ctl->qnt_h2o >= 0)
      if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
    if (ctl->qnt_o3 >= 0)
      if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
    if (ctl->qnt_theta >= 0)
      if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
    if (ctl->qnt_pv >= 0)
      if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));

    /* Check data... */
    for (ip = 0; ip < atm->np; ip++)
      if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
          || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
          || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
          || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
          || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
        atm->time[ip] = GSL_NAN;
        atm->p[ip] = GSL_NAN;
        atm->lon[ip] = GSL_NAN;
        atm->lat[ip] = GSL_NAN;
        for (iq = 0; iq < ctl->nq; iq++)
          atm->q[iq][ip] = GSL_NAN;
      } else {
        if (ctl->qnt_h2o >= 0)
          atm->q[ctl->qnt_h2o][ip] *= 1.608;
        if (ctl->qnt_pv >= 0)
          atm->q[ctl->qnt_pv][ip] *= 1e6;
        if (atm->lon[ip] > 180)
          atm->lon[ip] -= 360;
      }

    /* Close file... */
    NC(nc_close(ncid));
  }

  /* Error... */
  else
    ERRMSG("Atmospheric data type not supported!");

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

  /* Return success... */
  return 1;
}

```

```
/*****************************************************************************/

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

  int ip, iq;

  /* Write info... */
  printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
         "(executable: %s | compiled: %s, %s)\n\n",
         argv[0], __DATE__, __TIME__);

  /* Initialize quantity indices... */
  ctl->qnt_ens = -1;
  ctl->qnt_m = -1;
  ctl->qnt_r = -1;
  ctl->qnt_rho = -1;
  ctl->qnt_ps = -1;
  ctl->qnt_pt = -1;
  ctl->qnt_z = -1;
  ctl->qnt_p = -1;
  ctl->qnt_t = -1;
  ctl->qnt_u = -1;
  ctl->qnt_v = -1;
  ctl->qnt_w = -1;
  ctl->qnt_h2o = -1;
  ctl->qnt_o3 = -1;
  ctl->qnt_theta = -1;
  ctl->qnt_vh = -1;
  ctl->qnt_vz = -1;
  ctl->qnt_pv = -1;
  ctl->qnt_tice = -1;
  ctl->qnt_tsts = -1;
  ctl->qnt_tnat = -1;
  ctl->qnt_stat = -1;

  /* Read quantities... */
  ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
  if (ctl->nq > NQ)
    ERRMSG("Too many quantities!");
  for (iq = 0; iq < ctl->nq; iq++) {

    /* Read quantity name and format... */
    scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
    scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
             ctl->qnt_format[iq]);

    /* Try to identify quantity... */
    if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
      ctl->qnt_ens = iq;
      sprintf(ctl->qnt_unit[iq], "-");
    } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
      ctl->qnt_m = iq;
      sprintf(ctl->qnt_unit[iq], "kg");
    } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
      ctl->qnt_r = iq;
      sprintf(ctl->qnt_unit[iq], "m");
    } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
      ctl->qnt_rho = iq;
      sprintf(ctl->qnt_unit[iq], "kg/m^3");
    } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
      ctl->qnt_ps = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
      ctl->qnt_pt = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
      ctl->qnt_z = iq;
      sprintf(ctl->qnt_unit[iq], "km");
    } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
      ctl->qnt_p = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
      ctl->qnt_t = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
      ctl->qnt_u = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
      ctl->qnt_v = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
      ctl->qnt_w = iq;
      sprintf(ctl->qnt_unit[iq], "hPa/s");
    } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
      ctl->qnt_h2o = iq;
      sprintf(ctl->qnt_unit[iq], "1");
    } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
      ctl->qnt_o3 = iq;
      sprintf(ctl->qnt_unit[iq], "1");
    } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
      ctl->qnt_theta = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
      ctl->qnt_vh = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
      ctl->qnt_vz = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
      ctl->qnt_pv = iq;
      sprintf(ctl->qnt_unit[iq], "PVU");
    } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
      ctl->qnt_tice = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
      ctl->qnt_tsts = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
      ctl->qnt_tnat = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
      ctl->qnt_stat = iq;
      sprintf(ctl->qnt_unit[iq], "-");
    } else
      scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
  }

  /* Time steps of simulation... */
  ctl->direction =
    (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
  if (ctl->direction != -1 && ctl->direction != 1)
    ERRMSG("Set DIRECTION to -1 or 1!");
  ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
  ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);

  /* Meteorological data... */
  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
  ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
  ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
  ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
  ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
  ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
  ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
  ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
  if (ctl->met_np > EP)
    ERRMSG("Too many levels!");
  for (ip = 0; ip < ctl->met_np; ip++)
    ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
  ctl->met_tropo
    = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
  scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
  ctl->met_dt_out =
    scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);

  /* Isosurface parameters... */
  ctl->isosurf
    = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

  /* Diffusion parameters... */
  ctl->turb_dx_trop
    = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
  ctl->turb_dx_strat
    = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
  ctl->turb_dz_trop
    = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
  ctl->turb_dz_strat
    = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
  ctl->turb_mesox =
    scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
  ctl->turb_mesoz =
    scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);

  /* Mass and life time... */
  ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
  ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
  ctl->tdec_strat =
    scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);

  /* PSC analysis... */
  ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
  ctl->psc_hno3 =
    scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);

  /* Output of atmospheric data... */
  scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
  scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
  ctl->atm_dt_out =
    scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
  ctl->atm_filter =
    (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
  ctl->atm_type =
    (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);

  /* Output of CSI data... */
  scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
  ctl->csi_dt_out =
    scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
  scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
  ctl->csi_obsmin =
    scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
  ctl->csi_modmin =
    scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
  ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
  ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
  ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
  ctl->csi_lon0 =
    scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
  ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
  ctl->csi_nx =
    (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
  ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
  ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
  ctl->csi_ny =
    (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);

  /* Output of ensemble data... */
  scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);

  /* Output of grid data... */
  scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
           ctl->grid_basename);
  scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
  ctl->grid_dt_out =
    scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
  ctl->grid_sparse =
    (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
  ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
  ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
  ctl->grid_nz =
    (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
  ctl->grid_lon0 =
    scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
  ctl->grid_lon1 =
    scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
  ctl->grid_nx =
    (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
  ctl->grid_lat0 =
    scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
  ctl->grid_lat1 =
    scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
  ctl->grid_ny =
    (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);

  /* Output of profile data... */
  scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
           ctl->prof_basename);
  scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
  ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
  ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
  ctl->prof_nz =
    (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
  ctl->prof_lon0 =
    scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
  ctl->prof_lon1 =
    scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
  ctl->prof_nx =
    (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
  ctl->prof_lat0 =
    scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
  ctl->prof_lat1 =
    scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
  ctl->prof_ny =
    (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);

  /* Output of station data... */
  scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
           ctl->stat_basename);
  ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
  ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
  ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
}

/*****************************************************************************/

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

  char cmd[2 * LEN], levname[LEN], tstr[10];

  float help[EX * EY];

  int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;

  size_t np, nx, ny;

  /* Write info... */
  printf("Read meteorological data: %s\n", filename);

  /* Open netCDF file... */
  if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {

    /* Try to stage meteo file... */
    if (ctl->met_stage[0] != '-') {
      sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
              year, mon, day, hour, filename);
      if (system(cmd) != 0)
        ERRMSG("Error while staging meteo data!");
    }

    /* Try to open again... */
    if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
      return 0;
  }

  /* Get time from filename... */
  sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
  hour = atoi(tstr);
  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx < 2 || nx > EX)
    ERRMSG("Number of longitudes out of range!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");

  sprintf(levname, "lev");
  NC(nc_inq_dimid(ncid, levname, &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
  if (np == 1) {
    sprintf(levname, "lev_2");
    NC(nc_inq_dimid(ncid, levname, &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
  }
  if (np < 2 || np > EP)
    ERRMSG("Number of levels out of range!");

  /* Store dimensions... */
  met->np = (int) np;
  met->nx = (int) nx;
  met->ny = (int) ny;

  /* Get horizontal grid... */
  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));
  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

  /* Read meteorological data... */
  read_met_help(ncid, "t", "T", met, met->t, 1.0);
  read_met_help(ncid, "u", "U", met, met->u, 1.0);
  read_met_help(ncid, "v", "V", met, met->v, 1.0);
  read_met_help(ncid, "w", "W", met, met->w, 0.01f);
  read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
  read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));

  /* Meteo data on pressure levels... */
  if (ctl->met_np <= 0) {

    /* Read pressure levels from file... */
    NC(nc_inq_varid(ncid, levname, &varid));
    NC(nc_get_var_double(ncid, varid, met->p));
    for (ip = 0; ip < met->np; ip++)
      met->p[ip] /= 100.;

    /* Extrapolate data for lower boundary... */
    read_met_extrapolate(met);
  }

  /* Meteo data on model levels... */
  else {

    /* Read pressure data from file... */
    read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);

    /* Interpolate from model levels to pressure levels... */
    read_met_ml2pl(ctl, met, met->t);
    read_met_ml2pl(ctl, met, met->u);
    read_met_ml2pl(ctl, met, met->v);
    read_met_ml2pl(ctl, met, met->w);
    read_met_ml2pl(ctl, met, met->h2o);
    read_met_ml2pl(ctl, met, met->o3);

    /* Set pressure levels... */
    met->np = ctl->met_np;
    for (ip = 0; ip < met->np; ip++)
      met->p[ip] = ctl->met_p[ip];
  }

  /* Check ordering of pressure levels... */
  for (ip = 1; ip < met->np; ip++)
    if (met->p[ip - 1] < met->p[ip])
      ERRMSG("Pressure levels must be descending!");

  /* Read surface pressure... */
  if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
      || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
  } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
             || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
  } else
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->ps[ix][iy] = met->p[0];

  /* Create periodic boundary conditions... */
  read_met_periodic(met);

  /* Calculate geopotential heights... */
  read_met_geopot(ctl, met);

  /* Calculate potential vorticity... */
  read_met_pv(met);

  /* Calculate tropopause pressure... */
  read_met_tropo(ctl, met);

  /* Downsampling... */
  read_met_sample(ctl, met);

  /* Close file... */
  NC(nc_close(ncid));

  /* Return success... */
  return 1;
}

/*****************************************************************************/

void read_met_extrapolate(
  met_t * met) {

  int ip, ip0, ix, iy;

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      for (ip0 = met->np - 1; ip0 >= 0; ip0--)
        if (!gsl_finite(met->t[ix][iy][ip0])
            || !gsl_finite(met->u[ix][iy][ip0])
            || !gsl_finite(met->v[ix][iy][ip0])
            || !gsl_finite(met->w[ix][iy][ip0]))
          break;

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--) {
        met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
        met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
        met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
        met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
        met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
        met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
      }
    }
}

/*****************************************************************************/

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

  static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];

  static int init, topo_nx = -1, topo_ny;

  FILE *in;

  char line[LEN];

  double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;

  float help[EX][EY];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;

  /* Initialize geopotential heights... */
#pragma omp parallel for default(shared) private(ix,iy,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++)
        met->z[ix][iy][ip] = GSL_NAN;

  /* Check filename... */
  if (ctl->met_geopot[0] == '-')
    return;

  /* Read surface geopotential... */
  if (!init) {
    init = 1;

    /* Write info... */
    printf("Read surface geopotential: %s\n", ctl->met_geopot);

    /* Open file... */
    if (!(in = fopen(ctl->met_geopot, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
        if (rlon != rlon_old) {
          if ((++topo_nx) >= EX)
            ERRMSG("Too many longitudes!");
          topo_ny = 0;
        }
        rlon_old = rlon;
        topo_lon[topo_nx] = rlon;
        topo_lat[topo_ny] = rlat;
        topo_z[topo_nx][topo_ny] = rz;
        if ((++topo_ny) >= EY)
          ERRMSG("Too many latitudes!");
      }
    if ((++topo_nx) >= EX)
      ERRMSG("Too many longitudes!");

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

    /* Check grid spacing... */
    if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
        || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
      printf("Warning: Grid spacing does not match!\n");
  }

  /* Apply hydrostatic equation to calculate geopotential heights... */
#pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Get surface height... */
      lon = met->lon[ix];
      if (lon < topo_lon[0])
        lon += 360;
      else if (lon > topo_lon[topo_nx - 1])
        lon -= 360;
      lat = met->lat[iy];
      tx = locate_reg(topo_lon, topo_nx, lon);
      ty = locate_reg(topo_lat, topo_ny, lat);
      z0 = LIN(topo_lon[tx], topo_z[tx][ty],
               topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
      z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
               topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
      z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);

      /* Find surface pressure level... */
      ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);

      /* Get surface temperature... */
      ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
               met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
                   * log(met->ps[ix][iy] / met->p[ip0 + 1]));
      for (ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                     * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
                     * log(met->p[ip - 1] / met->p[ip]));
    }

  /* Smooth fields... */
  for (ip = 0; ip < met->np; ip++) {

    /* Median filter... */
#pragma omp parallel for default(shared) private(ix,iy,n,ix2,ix3,iy2,data)
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {
        n = 0;
        for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
               iy2++)
            if (gsl_finite(met->z[ix3][iy2][ip])) {
              data[n] = met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0) {
          gsl_sort(data, 1, (size_t) n);
          help[ix][iy] = (float)
            gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
        } else
          help[ix][iy] = GSL_NAN;
      }

    /* Copy data... */
#pragma omp parallel for default(shared) private(ix,iy)
    for (ix = 0; ix < met->nx; ix++)
```

```
      for (iy = 0; iy < met->ny; iy++)
        met->z[ix][iy][ip] = help[ix][iy];
  }
}

/*****************************************************************************/

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

  float *help;

  int ip, ix, iy, varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      return;

  /* Allocate... */
  ALLOC(help, float, met->nx * met->ny * met->np);

  /* Read data... */
  NC(nc_get_var_float(ncid, varid, help));

  /* Copy and check data... */
#pragma omp parallel for default(shared) private(ix,iy,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
        if (fabsf(dest[ix][iy][ip]) < 1e14f)
          dest[ix][iy][ip] *= scl;
        else
          dest[ix][iy][ip] = GSL_NAN;
      }

  /* Free... */
  free(help);
}

/*****************************************************************************/

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

  double aux[EP], p[EP], pt;

  int ip, ip2, ix, iy;

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Copy pressure profile... */
      for (ip = 0; ip < met->np; ip++)
        p[ip] = met->pl[ix][iy][ip];

      /* Interpolate... */
      for (ip = 0; ip < ctl->met_np; ip++) {
        pt = ctl->met_p[ip];
        if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
          pt = p[0];
        else if ((pt > p[met->np - 1] && p[1] > p[0])
                 || (pt < p[met->np - 1] && p[1] < p[0]))
          pt = p[met->np - 1];
        ip2 = locate_irr(p, met->np, pt);
        aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
                      p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
      }

      /* Copy data... */
      for (ip = 0; ip < ctl->met_np; ip++)
        var[ix][iy][ip] = (float) aux[ip];
    }
}

/*****************************************************************************/

```

```
void read_met_periodic(
  met_t * met) {

  int ip, iy;

  /* Check longitudes... */
  if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
             + met->lon[1] - met->lon[0] - 360) < 0.01))
    return;

  /* Increase longitude counter... */
  if ((++met->nx) > EX)
    ERRMSG("Cannot create periodic boundary conditions!");

  /* Set longitude... */
  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];

  /* Loop over latitudes and pressure levels... */
#pragma omp parallel for default(shared) private(iy,ip)
  for (iy = 0; iy < met->ny; iy++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->pt[met->nx - 1][iy] = met->pt[0][iy];
    for (ip = 0; ip < met->np; ip++) {
      met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
      met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
      met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
      met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
      met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
      met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
      met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
    }
  }
}

/*****************************************************************************/

void read_met_pv(
  met_t * met) {

  double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
    dtdp, dudp, dvdp, latr, vort, pows[EP];

  int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;

  /* Set powers... */
  for (ip = 0; ip < met->np; ip++)
    pows[ip] = pow(1000. / met->p[ip], 0.286);

  /* Loop over grid points... */
#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)
  for (ix = 0; ix < met->nx; ix++) {

    /* Set indices... */
    ix0 = GSL_MAX(ix - 1, 0);
    ix1 = GSL_MIN(ix + 1, met->nx - 1);

    /* Loop over grid points... */
    for (iy = 0; iy < met->ny; iy++) {

      /* Set indices... */
      iy0 = GSL_MAX(iy - 1, 0);
      iy1 = GSL_MIN(iy + 1, met->ny - 1);

      /* Set auxiliary variables... */
      latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
      dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
      dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
      c0 = cos(met->lat[iy0] / 180. * M_PI);
      c1 = cos(met->lat[iy1] / 180. * M_PI);
      cr = cos(latr / 180. * M_PI);
      vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);

      /* Loop over grid points... */
      for (ip = 0; ip < met->np; ip++) {

        /* Get gradients in longitude... */
        dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
        dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;

        /* Get gradients in latitude... */
        dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
        dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;

        /* Set indices... */
        ip0 = GSL_MAX(ip - 1, 0);
        ip1 = GSL_MIN(ip + 1, met->np - 1);

        /* Get gradients in pressure... */
        dp0 = 100. * (met->p[ip] - met->p[ip0]);
        dp1 = 100. * (met->p[ip1] - met->p[ip]);
        if (ip != ip0 && ip != ip1) {
          denom = dp0 * dp1 * (dp0 + dp1);
          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
                  - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
            / denom;
          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
                  - dp1 * dp1 * met->u[ix][iy][ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
            / denom;
          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
                  - dp1 * dp1 * met->v[ix][iy][ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
            / denom;
        } else {
          denom = dp0 + dp1;
          dtdp =
            (met->t[ix][iy][ip1] * pows[ip1] -
             met->t[ix][iy][ip0] * pows[ip0]) / denom;
          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
        }

        /* Calculate PV... */
        met->pv[ix][iy][ip] = (float)
          (1e6 * G0 *
           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
      }
    }
  }
}

/*****************************************************************************/

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

  met_t *help;

  float w, wsum;

  int ip, ip2, ix, ix2, ix3, iy, iy2;

  /* Check parameters... */
  if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
      && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
    return;

  /* Allocate... */
  ALLOC(help, met_t, 1);

  /* Copy data... */
  help->nx = met->nx;
  help->ny = met->ny;
  help->np = met->np;
  memcpy(help->lon, met->lon, sizeof(met->lon));
  memcpy(help->lat, met->lat, sizeof(met->lat));
  memcpy(help->p, met->p, sizeof(met->p));

  /* Smoothing... */
  for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
    for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
      for (ip = 0; ip < met->np; ip += ctl->met_dp) {
        help->ps[ix][iy] = 0;
        help->pt[ix][iy] = 0;
        help->z[ix][iy][ip] = 0;
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
        help->pv[ix][iy][ip] = 0;
        help->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        wsum = 0;
        for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
            for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
              w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
                * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
                * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->pt[ix][iy] += w * met->pt[ix3][iy2];
              help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
              help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
              help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
              help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
              help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
              help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
              help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->pt[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->z[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->pv[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Downsampling... */
  met->nx = 0;
  for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
      met->lat[met->ny] = help->lat[iy];
      met->ps[met->nx][met->ny] = help->ps[ix][iy];
      met->pt[met->nx][met->ny] = help->pt[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl->met_dp) {
        met->p[met->np] = help->p[ip];
        met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
        met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
        met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
        met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
        met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
        met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
        met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
        met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

  /* Free... */
  free(help);
}

/*****************************************************************************/

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

  gsl_interp_accel *acc;

  gsl_spline *spline;

  double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
    z[400], z2[400];

  int found, ix, iy, iz, iz2;

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);

  /* Get altitude and pressure profiles... */
  for (iz = 0; iz < met->np; iz++)
```

```
    z[iz] = Z(met->p[iz]);
  for (iz = 0; iz <= 170; iz++) {
    z2[iz] = 4.5 + 0.1 * iz;
    p2[iz] = P(z2[iz]);
  }

  /* Do not calculate tropopause... */
  if (ctl->met_tropo == 0)
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = GSL_NAN;

  /* Use tropopause climatology... */
  else if (ctl->met_tropo == 1)
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);

  /* Use cold point... */
  else if (ctl->met_tropo == 2) {

    /* Loop over grid points... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        gsl_spline_init(spline, z, t, (size_t) met->np);
        for (iz = 0; iz <= 170; iz++)
          t2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Find minimum... */
        iz = (int) gsl_stats_min_index(t2, 1, 171);
        if (iz <= 0 || iz >= 170)
          met->pt[ix][iy] = GSL_NAN;
        else
          met->pt[ix][iy] = p2[iz];
      }
  }

  /* Use WMO definition... */
  else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {

    /* Loop over grid points... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        gsl_spline_init(spline, z, t, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          t2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Find 1st tropopause... */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 140; iz++) {
          found = 1;
          for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
            if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                / log(p2[iz2] / p2[iz]) > 2.0) {
              found = 0;
              break;
            }
          if (found) {
            if (iz > 0 && iz < 140)
              met->pt[ix][iy] = p2[iz];
            break;
          }
        }

        /* Find 2nd tropopause... */
        if (ctl->met_tropo == 4) {
          met->pt[ix][iy] = GSL_NAN;
          for (; iz <= 140; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
              if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                  / log(p2[iz2] / p2[iz]) < 3.0) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 140; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
              if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                  / log(p2[iz2] / p2[iz]) > 2.0) {
                found = 0;
                break;
              }
            if (found) {
              if (iz > 0 && iz < 140)
                met->pt[ix][iy] = p2[iz];
              break;
            }
          }
        }
      }
  }

  /* Use dynamical tropopause... */
  else if (ctl->met_tropo == 5) {

    /* Loop over grid points... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {

        /* Interpolate potential vorticity profile... */
        for (iz = 0; iz < met->np; iz++)
          pv[iz] = met->pv[ix][iy][iz];
        gsl_spline_init(spline, z, pv, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Interpolate potential temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
        gsl_spline_init(spline, z, th, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          th2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Find dynamical tropopause 3.5 PVU + 380 K */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 160; iz++)
          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
            if (iz > 0 && iz < 160)
              met->pt[ix][iy] = p2[iz];
            break;
          }
      }
  }

  else
    ERRMSG("Cannot calculate tropopause!");

  /* Free... */
  gsl_spline_free(spline);
  gsl_interp_accel_free(acc);
}

/*****************************************************************************/

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

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    msg[2 * LEN], rvarname[LEN], rval[LEN];

  int contain = 0, i;

  /* Open file... */
  if (filename[strlen(filename) - 1] != '-')
    if (!(in = fopen(filename, "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
        }
  for (i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else {
      sprintf(msg, "Missing variable %s!\n", fullname1);
      ERRMSG(msg);
    }
  }

  /* Write info... */
  printf("%s = %s\n", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}

/*****************************************************************************/

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

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}

/*****************************************************************************/

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

  static double starttime[NTIMER], runtime[NTIMER];

  /* Check id... */
  if (id < 0 || id >= NTIMER)
    ERRMSG("Too many timers!");

  /* Start timer... */
  if (mode == 1) {
    if (starttime[id] <= 0)
      starttime[id] = omp_get_wtime();
    else
      ERRMSG("Timer already started!");
  }

  /* Stop timer... */
  else if (mode == 2) {
    if (starttime[id] > 0) {
      runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
      starttime[id] = -1;
    }
  }

  /* Print timer... */
  else if (mode == 3) {
    printf("%s = %.3f s\n", name, runtime[id]);
    runtime[id] = 0;
  }
}

/*****************************************************************************/

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

  FILE *in, *out;

  char line[LEN];

  double r, t0, t1;

  int ip, iq, year, mon, day, hour, min, sec;

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Write info... */
  printf("Write atmospheric data: %s\n", filename);

  /* Write ASCII data... */
  if (ctl->atm_type == 0) {

    /* Check if gnuplot output is requested... */
    if (ctl->atm_gpfile[0] != '-') {

      /* Create gnuplot pipe... */
      if (!(out = popen("gnuplot", "w")))
        ERRMSG("Cannot create pipe to gnuplot!");

      /* Set plot filename... */
      fprintf(out, "set out \"%s.png\"\n", filename);

      /* Set time string... */
      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
      fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
              year, mon, day, hour, min);

      /* Dump gnuplot file to pipe... */
      if (!(in = fopen(ctl->atm_gpfile, "r")))
        ERRMSG("Cannot open file!");
      while (fgets(line, LEN, in))
        fprintf(out, "%s", line);
      fclose(in);
    }

    else {

      /* Create file... */
      if (!(out = fopen(filename, "w")))
        ERRMSG("Cannot create file!");
    }

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
              ctl->qnt_unit[iq]);
    fprintf(out, "\n");

    /* Write data... */
    for (ip = 0; ip < atm->np; ip++) {

      /* Check time... */
      if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
        continue;

      /* Write output... */
      fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
              atm->lon[ip], atm->lat[ip]);
      for (iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
      }
      fprintf(out, "\n");
    }

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

  /* Write binary data... */
  else if (ctl->atm_type == 1) {

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write data... */
    FWRITE(&atm->np, int,
           1,
           out);
    FWRITE(atm->time, double,
             (size_t) atm->np,
           out);
    FWRITE(atm->p, double,
             (size_t) atm->np,
           out);
    FWRITE(atm->lon, double,
             (size_t) atm->np,
           out);
    FWRITE(atm->lat, double,
             (size_t) atm->np,
           out);
    for (iq = 0; iq < ctl->nq; iq++)
      FWRITE(atm->q[iq], double,
               (size_t) atm->np,
             out);

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

  /* Error... */
  else
    ERRMSG("Atmospheric data type not supported!");
}

/*****************************************************************************/

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

  static FILE *in, *out;

  static char line[LEN];

  static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
    rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;

  static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantity index for mass... */
    if (ctl->qnt_m < 0)
      ERRMSG("Need quantity mass!");

    /* Open observation data file... */
    printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
    if (!(in = fopen(ctl->csi_obsfile, "r")))
      ERRMSG("Cannot open file!");

    /* Create new file... */
    printf("Write CSI data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = number of hits (cx)\n"
            "# $3 = number of misses (cy)\n"
            "# $4 = number of false alarms (cz)\n"
            "# $5 = number of observations (cx + cy)\n"
            "# $6 = number of forecasts (cx + cz)\n"
            "# $7 = bias (forecasts/observations) [%%]\n"
            "# $8 = probability of detection (POD) [%%]\n"
            "# $9 = false alarm rate (FAR) [%%]\n"
            "# $10 = critical success index (CSI) [%%]\n\n");
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Initialize grid cells... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->csi_nx; ix++)
    for (iy = 0; iy < ctl->csi_ny; iy++)
      for (iz = 0; iz < ctl->csi_nz; iz++)
        modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;

  /* Read observation data... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
        5)
      continue;

    /* Check time... */
    if (rt < t0)
      continue;
    if (rt > t1)
      break;

    /* Calculate indices... */
    ix = (int) ((rlon - ctl->csi_lon0)
                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
    iy = (int) ((rlat - ctl->csi_lat0)
                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
    iz = (int) ((rz - ctl->csi_z0)
                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->csi_nx ||
        iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
      continue;

    /* Get mean observation index... */
    obsmean[ix][iy][iz] += robs;
    obscount[ix][iy][iz]++;
  }

  /* Analyze model data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Get indices... */
    ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
    iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
    iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->csi_nx ||
        iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
      continue;

    /* Get total mass in grid cell... */
    modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  }

  /* Analyze all grid cells... */
#pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
  for (ix = 0; ix < ctl->csi_nx; ix++)
    for (iy = 0; iy < ctl->csi_ny; iy++)
      for (iz = 0; iz < ctl->csi_nz; iz++) {

        /* Calculate mean observation index... */
        if (obscount[ix][iy][iz] > 0)
          obsmean[ix][iy][iz] /= obscount[ix][iy][iz];

        /* Calculate column density... */
        if (modmean[ix][iy][iz] > 0) {
          dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
          dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
          lat = ctl->csi_lat0 + dlat * (iy + 0.5);
          area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
            * cos(lat * M_PI / 180.);
          modmean[ix][iy][iz] /= (1e6 * area);
        }

        /* Calculate CSI... */
        if (obscount[ix][iy][iz] > 0) {
          if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
              modmean[ix][iy][iz] >= ctl->csi_modmin)
            cx++;
          else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                   modmean[ix][iy][iz] < ctl->csi_modmin)
            cy++;
          else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                   modmean[ix][iy][iz] >= ctl->csi_modmin)
            cz++;
        }
      }

  /* Write output... */
  if (fmod(t, ctl->csi_dt_out) == 0) {

    /* Write... */
    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
            t, cx, cy, cz, cx + cy, cx + cz,
            (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
            (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
            (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
            (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);

    /* Set counters to zero... */
    cx = cy = cz = 0;
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}

/*****************************************************************************/

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

  static FILE *out;

  static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
    t0, t1, x[NENS][3], xm[3];

  static int ip, iq;

  static size_t i, n;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantities... */
    if (ctl->qnt_ens < 0)
      ERRMSG("Missing ensemble IDs!");

    /* Create new file... */
    printf("Write ensemble data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Init... */
  ens = GSL_NAN;
  n = 0;

  /* Loop over air parcels... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Check ensemble id... */
    if (atm->q[ctl->qnt_ens][ip] != ens) {

      /* Write results... */
      if (n > 0) {

        /* Get mean position... */
        xm[0] = xm[1] = xm[2] = 0;
        for (i = 0; i < n; i++) {
          xm[0] += x[i][0] / (double) n;
          xm[1] += x[i][1] / (double) n;
          xm[2] += x[i][2] / (double) n;
        }
        cart2geo(xm, &dummy, &lon, &lat);
        fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
                lat);

        /* Get quantity statistics... */
        for (iq = 0; iq < ctl->nq; iq++) {
          fprintf(out, " ");
          fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
        }
        for (iq = 0; iq < ctl->nq; iq++) {
          fprintf(out, " ");
          fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
        }
        fprintf(out, " %lu\n", n);
      }

      /* Init new ensemble... */
      ens = atm->q[ctl->qnt_ens][ip];
      n = 0;
    }

    /* Save data... */
    p[n] = atm->p[ip];
    geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
    for (iq = 0; iq < ctl->nq; iq++)
      q[iq][n] = atm->q[iq][ip];
    if ((++n) >= NENS)
      ERRMSG("Too many data points!");
  }

  /* Write results... */
  if (n > 0) {

    /* Get mean position... */
    xm[0] = xm[1] = xm[2] = 0;
    for (i = 0; i < n; i++) {
      xm[0] += x[i][0] / (double) n;
      xm[1] += x[i][1] / (double) n;
      xm[2] += x[i][2] / (double) n;
    }
    cart2geo(xm, &dummy, &lon, &lat);
    fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);

    /* Get quantity statistics... */
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
    }
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
    }
    fprintf(out, " %lu\n", n);
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}

/*****************************************************************************/

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

  FILE *in, *out;

  char line[LEN];

  static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
    area, rho_air, press, temp, cd, vmr, t0, t1, r;

  static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;

  /* Check dimensions... */
  if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Set grid box size... */
  dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
  dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
  dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;

  /* Initialize grid... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->grid_nx; ix++)
    for (iy = 0; iy < ctl->grid_ny; iy++)
      for (iz = 0; iz < ctl->grid_nz; iz++) {
        mass[ix][iy][iz] = 0;
        np[ix][iy][iz] = 0;
      }

  /* Average data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++)
    if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {

      /* Get index... */
      ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
      iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
      iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);

      /* Check indices... */
      if (ix < 0 || ix >= ctl->grid_nx ||
          iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
        continue;

      /* Add mass... */
      if (ctl->qnt_m >= 0)
        mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
      np[ix][iy][iz]++;
    }

  /* Check if gnuplot output is requested... */
  if (ctl->grid_gpfile[0] != '-') {

    /* Write info... */
    printf("Plot grid data: %s.png\n", filename);

    /* Create gnuplot pipe... */
    if (!(out = popen("gnuplot", "w")))
      ERRMSG("Cannot create pipe to gnuplot!");

    /* Set plot filename... */
    fprintf(out, "set out \"%s.png\"\n", filename);

    /* Set time string... */
    jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
    fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
            year, mon, day, hour, min);

    /* Dump gnuplot file to pipe... */
    if (!(in = fopen(ctl->grid_gpfile, "r")))
      ERRMSG("Cannot open file!");
    while (fgets(line, LEN, in))
      fprintf(out, "%s", line);
    fclose(in);
  }

  else {

    /* Write info... */
    printf("Write grid data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");
  }

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = surface area [km^2]\n"
          "# $6 = layer width [km]\n"
          "# $7 = number of particles [1]\n"
          "# $8 = column density [kg/m^2]\n"
          "# $9 = volume mixing ratio [1]\n\n");

  /* Write data... */
  for (ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
      fprintf(out, "\n");
    for (iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
        fprintf(out, "\n");
      for (iz = 0; iz < ctl->grid_nz; iz++)
        if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {

          /* Set coordinates... */
          z = ctl->grid_z0 + dz * (iz + 0.5);
          lon = ctl->grid_lon0 + dlon * (ix + 0.5);
          lat = ctl->grid_lat0 + dlat * (iy + 0.5);

          /* Get pressure and temperature... */
          press = P(z);
          intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                          NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);

          /* Calculate surface area... */
          area = dlat * dlon * SQR(RE * M_PI / 180.)
            * cos(lat * M_PI / 180.);

          /* Calculate column density... */
          cd = mass[ix][iy][iz] / (1e6 * area);

          /* Calculate volume mixing ratio... */
          rho_air = 100. * press / (RA * temp);
          vmr = MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * 1e6 * area * 1e3 * dz);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
                  t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
        }
    }
  }

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

/*****************************************************************************/

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

```

  static FILE *in, *out;

  static char line[LEN];

  static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
    rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
    rho_air, vmr, h2o, o3;

  static int obscount[GX][GY], ip, ix, iy, iz, okay;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantity index for mass... */
    if (ctl->qnt_m < 0)
      ERRMSG("Need quantity mass!");

    /* Check dimensions... */
    if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
      ERRMSG("Grid dimensions too large!");

    /* Open observation data file... */
    printf("Read profile observation data: %s\n", ctl->prof_obsfile);
    if (!(in = fopen(ctl->prof_obsfile, "r")))
      ERRMSG("Cannot open file!");

    /* Create new output file... */
    printf("Write profile data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n"
            "# $4 = latitude [deg]\n"
            "# $5 = pressure [hPa]\n"
            "# $6 = temperature [K]\n"
            "# $7 = volume mixing ratio [1]\n"
            "# $8 = H2O volume mixing ratio [1]\n"
            "# $9 = O3 volume mixing ratio [1]\n"
            "# $10 = observed BT index (mean) [K]\n"
            "# $11 = observed BT index (sigma) [K]\n");

    /* Set grid box size... */
    dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
    dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
    dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Initialize... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++) {
      obsmean[ix][iy] = 0;
      obsmean2[ix][iy] = 0;
      obscount[ix][iy] = 0;
      for (iz = 0; iz < ctl->prof_nz; iz++)
        mass[ix][iy][iz] = 0;
    }

  /* Read observation data... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
        5)
      continue;

    /* Check time... */
    if (rt < t0)
      continue;
    if (rt > t1)
      break;

    /* Calculate indices... */
    ix = (int) ((rlon - ctl->prof_lon0) / dlon);
    iy = (int) ((rlat - ctl->prof_lat0) / dlat);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
```

```
      continue;

    /* Get mean observation index... */
    obsmean[ix][iy] += robs;
    obsmean2[ix][iy] += SQR(robs);
    obscount[ix][iy]++;
  }

  /* Analyze model data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Get indices... */
    ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
    iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
    iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->prof_nx ||
        iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
      continue;

    /* Get total mass in grid cell... */
    mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  }

  /* Extract profiles... */
  for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++)
      if (obscount[ix][iy] > 0) {

        /* Check profile... */
        okay = 0;
        for (iz = 0; iz < ctl->prof_nz; iz++)
          if (mass[ix][iy][iz] > 0) {
            okay = 1;
            break;
          }
        if (!okay)
          continue;

        /* Write output... */
        fprintf(out, "\n");

        /* Loop over altitudes... */
        for (iz = 0; iz < ctl->prof_nz; iz++) {

          /* Set coordinates... */
          z = ctl->prof_z0 + dz * (iz + 0.5);
          lon = ctl->prof_lon0 + dlon * (ix + 0.5);
          lat = ctl->prof_lat0 + dlat * (iy + 0.5);

          /* Get pressure and temperature... */
          press = P(z);
          intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                          NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);

          /* Calculate surface area... */
          area = dlat * dlon * SQR(M_PI * RE / 180.)
            * cos(lat * M_PI / 180.);

          /* Calculate volume mixing ratio... */
          rho_air = 100. * press / (RA * temp);
          vmr = MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * area * dz * 1e9);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
                  t, z, lon, lat, press, temp, vmr, h2o, o3,
                  obsmean[ix][iy] / obscount[ix][iy],
                  sqrt(obsmean2[ix][iy] / obscount[ix][iy]
                       - SQR(obsmean[ix][iy] / obscount[ix][iy])));
        }
      }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}

/*****************************************************************************/

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

  static FILE *out;

  static double rmax2, t0, t1, x0[3], x1[3];

  static int ip, iq;

  /* Init... */
  if (t == ctl->t_start) {

    /* Write info... */
    printf("Write station data: %s\n", filename);

    /* Create new file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    fprintf(out, "\n");

    /* Set geolocation and search radius... */
    geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
    rmax2 = SQR(ctl->stat_r);
  }

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Loop over air parcels... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Check station flag... */
    if (ctl->qnt_stat >= 0)
      if (atm->q[ctl->qnt_stat][ip])
        continue;

    /* Get Cartesian coordinates... */
    geo2cart(0, atm->lon[ip], atm->lat[ip], x1);

    /* Check horizontal distance... */
    if (DIST2(x0, x1) > rmax2)
      continue;

    /* Set station flag... */
    if (ctl->qnt_stat >= 0)
      atm->q[ctl->qnt_stat][ip] = 1;

    /* Write data... */
    fprintf(out, "%.2f %g %g %g",
            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

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.

```
               {

  double radius;

  radius = NORM(x);
  *lat = asin(x[2] / radius) * 180 / M_PI;
  *lon = atan2(x[1], x[0]) * 180 / M_PI;
  *z = radius - RE;
}
```

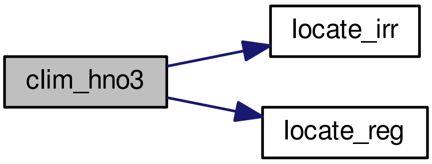
5.21.2.2 double clim_hno3 (double *t*, double *lat*, double *p*)

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},
   {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
   {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
   {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
   {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
   {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
   {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
   {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
   {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
   {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
   {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
   {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
   {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
  {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
   {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
   {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
   {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
   {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
   {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
   {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
   {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
   {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
   {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
   {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
   {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
   {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
   {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
   {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
   {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
   {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
   {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
  {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
   {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
   {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
   {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
   {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
   {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
   {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
   {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
   {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
   {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
   {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
   {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
   {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
   {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
   {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
   {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
   {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
   {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
  {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
   {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
   {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
   {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
   {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
   {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
   {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
   {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
   {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
   {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
   {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
   {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
   {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
   {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
   {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
   {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
   {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
   {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
  {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
   {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
   {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
   {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
   {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
   {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
   {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
   {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
   {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
   {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
   {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
   {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
   {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
   {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
   {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
   {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
   {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
   {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
  {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
   {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
   {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
   {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
   {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
   {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
   {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
   {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
   {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
   {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
   {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
   {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
   {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
   {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
   {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
   {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
   {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
   {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
  {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
   {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
   {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
   {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
   {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
   {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
   {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
   {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
   {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
   {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
   {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
   {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
   {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
   {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
   {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
   {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
   {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
   {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
  {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
   {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
   {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
   {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
   {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
   {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
   {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
   {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
   {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
   {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
   {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
   {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
   {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
   {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
   {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
   {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
   {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
   {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
  {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
   {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
   {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
   {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
   {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
   {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
   {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
   {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
   {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
   {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
   {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
   {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
   {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
   {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
   {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
   {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
   {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
   {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
};

double aux00, aux01, aux10, aux11, sec;

int ilat, ip, isec;

/* Get seconds since begin of year... */
sec = fmod(t, 365.25 * 86400.);

/* Get indices... */
isec = locate_irr(secs, 12, sec);
ilat = locate_reg(lats, 18, lat);
ip = locate_irr(ps, 10, p);

/* Interpolate... */
aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
            ps[ip + 1], hno3[isec][ilat][ip + 1], p);
aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
            ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
            ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
            ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
}
```

Here is the call graph for this function:



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

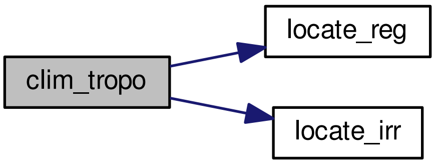
Climatology of tropopause pressure.

Definition at line 311 of file libtrac.c.

```
              {

  static double doys[12]
  = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };

  static double lats[73]
    = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
    -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
    -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
    -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
    15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
    45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
    75, 77.5, 80, 82.5, 85, 87.5, 90
  };

  static double tps[12][73]
    = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
         297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
         175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
         99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
         98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
         152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
         277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
         275.3, 275.6, 275.4, 274.1, 273.5},
  {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
   300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
   150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
   98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
   98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
   220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
   284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
   287.5, 286.2, 285.8},
  {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
   297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
   161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
   100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
   99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
   186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
   279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
   304.3, 304.9, 306, 306.6, 306.2, 306},
{306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
 290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
 195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
 102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
 99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
 148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
 263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
 315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
{266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
 260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
 205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
 101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
 102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
 165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
 273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
 325.3, 325.8, 325.8},
{220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
 222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
 228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
 105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
 106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
 127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
 251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
 308.5, 312.2, 313.1, 313.3},
{187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
 187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
 235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
 110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
 111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
 117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
 224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
 275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
{166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
 185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
 233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
 110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
 112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
 120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
 230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
 278.2, 282.6, 287.4, 290.9, 292.5, 293},
{171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
 183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
 243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
 114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
 110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
 114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
 203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
 276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
{191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
 215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
 237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
 111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
 106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
 112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
 206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
 279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
 305.1},
{241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
{301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
 281.7, 281.1, 281.2}
};

double doy, p0, p1;

int imon, ilat;

/* Get day of year... */
doy = fmod(t / 86400., 365.25);
while (doy < 0)
  doy += 365.25;

/* Get indices... */
ilat = locate_reg(lats, 73, lat);
imon = locate_irr(doys, 12, doy);
```

```

  /* Interpolate... */
  p0 = LIN(lats[ilat], tps[imon][ilat],
           lats[ilat + 1], tps[imon][ilat + 1], lat);
  p1 = LIN(lats[ilat], tps[imon + 1][ilat],
           lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
  return LIN(doys[imon], p0, doys[imon + 1], p1, doy);
}
```

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.

```
            {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

  /* Get day of year... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
    *doy = d01[mon - 1] + day - 1;
  else
    *doy = d0[mon - 1] + day - 1;
}
```

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.

```
            {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
  int i;

  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d01[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d01[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}
```

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.

```
             {

  double radius;

  radius = z + RE;
  x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
  x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
  x[2] = radius * sin(lat / 180 * M_PI);
}
```

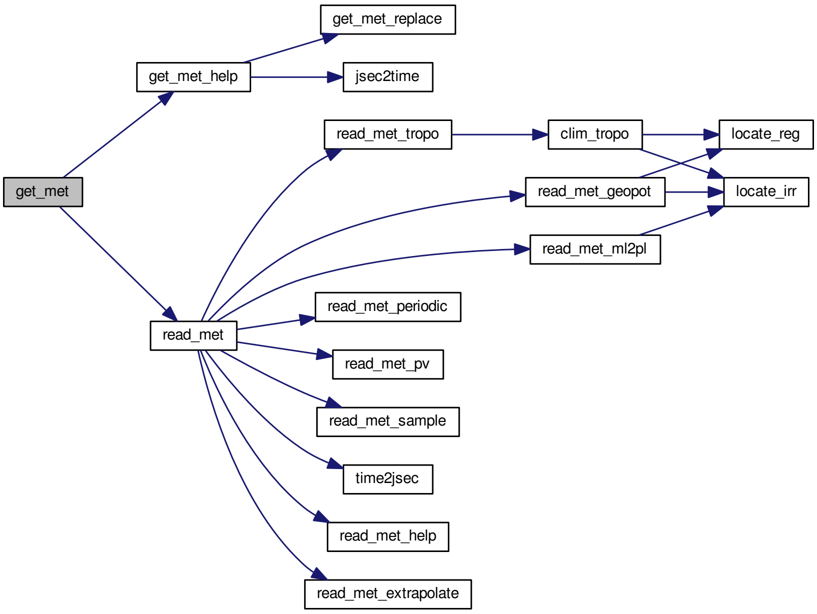
5.21.2.7 void get_met (ctl_t ∗ *ctl*, char ∗ *metbase*, double *t*, met_t ∗∗ *met0*, met_t ∗∗ *met1*)

Get meteorological data for given timestep.

Definition at line 513 of file libtrac.c.

```
                 {

  static int init, ip, ix, iy;

  met_t *mets;

  char filename[LEN];

  /* Init... */
  if (t == ctl->t_start || !init) {
    init = 1;

    get_met_help(t, -1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met0))
      ERRMSG("Cannot open file!");

    get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met1))
      ERRMSG("Cannot open file!");
  }

  /* Read new data for forward trajectories... */
  if (t > (*met1)->time && ctl->direction == 1) {
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;
    get_met_help(t, 1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met1))
      ERRMSG("Cannot open file!");
  }

  /* Read new data for backward trajectories... */
  if (t < (*met0)->time && ctl->direction == -1) {
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;
    get_met_help(t, -1, metbase, ctl->dt_met, filename);
    if (!read_met(ctl, filename, *met0))
      ERRMSG("Cannot open file!");
  }

  /* Check that grids are consistent... */
  if ((*met0)->nx != (*met1)->nx
      || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
    ERRMSG("Meteo grid dimensions do not match!");
  for (ix = 0; ix < (*met0)->nx; ix++)
    if ((*met0)->lon[ix] != (*met1)->lon[ix])
      ERRMSG("Meteo grid longitudes do not match!");
  for (iy = 0; iy < (*met0)->ny; iy++)
    if ((*met0)->lat[iy] != (*met1)->lat[iy])
      ERRMSG("Meteo grid latitudes do not match!");
  for (ip = 0; ip < (*met0)->np; ip++)
    if ((*met0)->p[ip] != (*met1)->p[ip])
      ERRMSG("Meteo grid pressure levels do not match!");
}
```

Here is the call graph for this function:



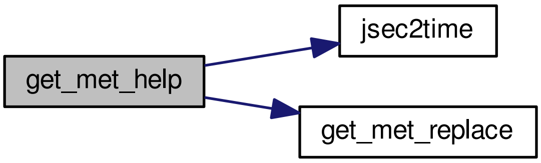
5.21.2.8 void get_met_help (double *t*, int *direct*, char ∗ *metbase*, double *dt_met*, char ∗ *filename*)

Get meteorological data for timestep.

Definition at line 576 of file libtrac.c.

```
                    {

  char repl[LEN];

  double t6, r;

  int year, mon, day, hour, min, sec;

  /* Round time to fixed intervals... */
  if (direct == -1)
    t6 = floor(t / dt_met) * dt_met;
  else
    t6 = ceil(t / dt_met) * dt_met;

  /* Decode time... */
  jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Set filename... */
  sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
  sprintf(repl, "%d", year);
  get_met_replace(filename, "YYYY", repl);
  sprintf(repl, "%02d", mon);
  get_met_replace(filename, "MM", repl);
  sprintf(repl, "%02d", day);
  get_met_replace(filename, "DD", repl);
  sprintf(repl, "%02d", hour);
  get_met_replace(filename, "HH", repl);
}
```

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.

```
                {

  char buffer[LEN], *ch;

  int i;

  /* Iterate... */
  for (i = 0; i < 3; i++) {

    /* Replace substring... */
    if (!(ch = strstr(orig, search)))
      return;
    strncpy(buffer, orig, (size_t) (ch - orig));
    buffer[ch - orig] = 0;
    sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
    orig[0] = 0;
    strcpy(orig, buffer);
  }
}
```

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.

```
                 {

  double aux00, aux01, aux10, aux11;

  /* Set variables... */
  aux00 = array[ix][iy];
  aux01 = array[ix][iy + 1];
  aux10 = array[ix + 1][iy];
  aux11 = array[ix + 1][iy + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}
```

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.

```
               {

  double aux00, aux01, aux10, aux11;

  /* Interpolate vertically... */
  aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
    + array[ix][iy][ip + 1];
  aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
    + array[ix][iy + 1][ip + 1];
  aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
    + array[ix + 1][iy][ip + 1];
  aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
    + array[ix + 1][iy + 1][ip + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}
```

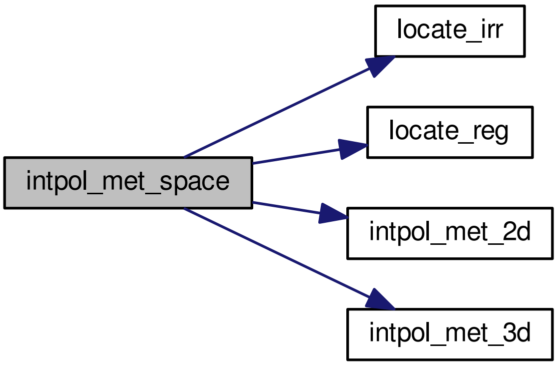
5.21.2.12 void intpol_met_space (met_t * *met,* double *p,* double *lon,* double *lat,* double * *ps,* double * *pt,* double * *z,* double * *t,* double * *u,* double * *v,* double * *w,* double * *pv,* double * *h2o,* double * *o3*)

Spatial interpolation of meteorological data.

Definition at line 691 of file libtrac.c.

```
              {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (met->lon[met->nx - 1] > 180 && lon < 0)
    lon += 360;

  /* Get indices... */
  ip = locate_irr(met->p, met->np, p);
  ix = locate_reg(met->lon, met->nx, lon);
  iy = locate_reg(met->lat, met->ny, lat);

  /* Get weights... */
  wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
  wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
  wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);

  /* Interpolate... */
  if (ps != NULL)
    intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
  if (pt != NULL)
    intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
  if (z != NULL)
    intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
  if (t != NULL)
    intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
  if (u != NULL)
    intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
  if (v != NULL)
    intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
  if (w != NULL)
    intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
  if (pv != NULL)
    intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
  if (h2o != NULL)
    intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
  if (o3 != NULL)
    intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
}
```

Here is the call graph for this function:



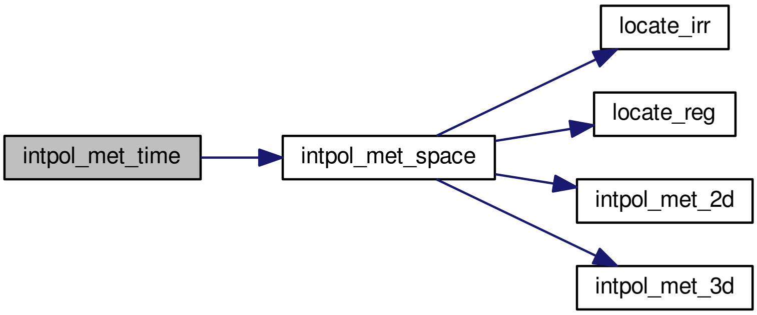
5.21.2.13 void intpol_met_time (met_t * *met0,* met_t * *met1,* double *ts,* double *p,* double *lon,* double *lat,* double * *ps,* double * *pt,* double * *z,* double * *t,* double * *u,* double * *v,* double * *w,* double * *pv,* double * *h2o,* double * *o3*)

Temporal interpolation of meteorological data.

Definition at line 750 of file libtrac.c.

```
            {

double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
  v0, v1, w0, w1, wt, z0, z1;

/* Spatial interpolation... */
intpol_met_space(met0, p, lon, lat,
                 ps == NULL ? NULL : &ps0,
                 pt == NULL ? NULL : &pt0,
                 z == NULL ? NULL : &z0,
                 t == NULL ? NULL : &t0,
                 u == NULL ? NULL : &u0,
                 v == NULL ? NULL : &v0,
                 w == NULL ? NULL : &w0,
                 pv == NULL ? NULL : &pv0,
                 h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
intpol_met_space(met1, p, lon, lat,
                 ps == NULL ? NULL : &ps1,
                 pt == NULL ? NULL : &pt1,
                 z == NULL ? NULL : &z1,
                 t == NULL ? NULL : &t1,
                 u == NULL ? NULL : &u1,
                 v == NULL ? NULL : &v1,
                 w == NULL ? NULL : &w1,
                 pv == NULL ? NULL : &pv1,
                 h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);

/* Get weighting factor... */
wt = (met1->time - ts) / (met1->time - met0->time);

/* Interpolate... */
if (ps != NULL)
  *ps = wt * (ps0 - ps1) + ps1;
if (pt != NULL)
  *pt = wt * (pt0 - pt1) + pt1;
if (z != NULL)
  *z = wt * (z0 - z1) + z1;
if (t != NULL)
  *t = wt * (t0 - t1) + t1;
if (u != NULL)
  *u = wt * (u0 - u1) + u1;
  if (v != NULL)
    *v = wt * (v0 - v1) + v1;
  if (w != NULL)
    *w = wt * (w0 - w1) + w1;
  if (pv != NULL)
    *pv = wt * (pv0 - pv1) + pv1;
  if (h2o != NULL)
    *h2o = wt * (h2o0 - h2o1) + h2o1;
  if (o3 != NULL)
    *o3 = wt * (o30 - o31) + o31;
}
```

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.

```
                {

  struct tm t0, *t1;

  time_t jsec0;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}
```

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.

```
            {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}
```

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.

```
            {

  int i;

  /* Calculate index... */
  i = (int) ((x - xx[0]) / (xx[1] - xx[0]));

  /* Check range... */
  if (i < 0)
    i = 0;
  else if (i >= n - 2)
    i = n - 2;

  return i;
}
```

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.

```
             {

FILE *in;

char line[LEN], *tok;

double t0;

int dimid, ip, iq, ncid, varid;

size_t nparts;

/* Init... */
atm->np = 0;

/* Write info... */
printf("Read atmospheric data: %s\n", filename);

/* Read ASCII data... */
if (ctl->atm_type == 0) {

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    return 0;

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    for (iq = 0; iq < ctl->nq; iq++)
      TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

    /* Convert altitude to pressure... */
    atm->p[atm->np] = P(atm->p[atm->np]);

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

  /* Close file... */
  fclose(in);
}

/* Read binary data... */
else if (ctl->atm_type == 1) {

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    return 0;

  /* Read data... */
  FREAD(&atm->np, int, 1, in);
  FREAD(atm->time, double,
          (size_t) atm->np,
        in);
  FREAD(atm->p, double,
          (size_t) atm->np,
        in);
  FREAD(atm->lon, double,
          (size_t) atm->np,
        in);
  FREAD(atm->lat, double,
          (size_t) atm->np,
        in);
  for (iq = 0; iq < ctl->nq; iq++)
    FREAD(atm->q[iq], double,
            (size_t) atm->np,
          in);

  /* Close file... */
  fclose(in);
}

/* Read netCDF data... */
else if (ctl->atm_type == 2) {

  /* Open file... */
  if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
    return 0;

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
```

```
    NC(nc_inq_dimlen(ncid, dimid, &nparts));
    atm->np = (int) nparts;
    if (atm->np > NP)
      ERRMSG("Too many particles!");

    /* Get time... */
    NC(nc_inq_varid(ncid, "time", &varid));
    NC(nc_get_var_double(ncid, varid, &t0));
    for (ip = 0; ip < atm->np; ip++)
      atm->time[ip] = t0;

    /* Read geolocations... */
    NC(nc_inq_varid(ncid, "PRESS", &varid));
    NC(nc_get_var_double(ncid, varid, atm->p));
    NC(nc_inq_varid(ncid, "LON", &varid));
    NC(nc_get_var_double(ncid, varid, atm->lon));
    NC(nc_inq_varid(ncid, "LAT", &varid));
    NC(nc_get_var_double(ncid, varid, atm->lat));

    /* Read variables... */
    if (ctl->qnt_p >= 0)
      if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
    if (ctl->qnt_t >= 0)
      if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
    if (ctl->qnt_u >= 0)
      if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
    if (ctl->qnt_v >= 0)
      if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
    if (ctl->qnt_w >= 0)
      if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
    if (ctl->qnt_h2o >= 0)
      if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
    if (ctl->qnt_o3 >= 0)
      if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
    if (ctl->qnt_theta >= 0)
      if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
    if (ctl->qnt_pv >= 0)
      if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
        NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));

    /* Check data... */
    for (ip = 0; ip < atm->np; ip++)
      if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
          || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
          || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
          || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
          || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
        atm->time[ip] = GSL_NAN;
        atm->p[ip] = GSL_NAN;
        atm->lon[ip] = GSL_NAN;
        atm->lat[ip] = GSL_NAN;
        for (iq = 0; iq < ctl->nq; iq++)
          atm->q[iq][ip] = GSL_NAN;
      } else {
        if (ctl->qnt_h2o >= 0)
          atm->q[ctl->qnt_h2o][ip] *= 1.608;
        if (ctl->qnt_pv >= 0)
          atm->q[ctl->qnt_pv][ip] *= 1e6;
        if (atm->lon[ip] > 180)
          atm->lon[ip] -= 360;
      }

    /* Close file... */
    NC(nc_close(ncid));
  }

  /* Error... */
  else
    ERRMSG("Atmospheric data type not supported!");

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

  /* Return success... */
  return 1;
}
```

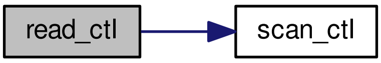
5.21.2.18 void read_ctl (const char ∗ *filename,* int *argc,* char ∗ *argv[ ],* ctl_t ∗ *ctl*)

Read control parameters.

Definition at line 1087 of file libtrac.c.

```
               {

  int ip, iq;

  /* Write info... */
  printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
         "(executable: %s | compiled: %s, %s)\n\n",
         argv[0], __DATE__, __TIME__);

  /* Initialize quantity indices... */
  ctl->qnt_ens = -1;
  ctl->qnt_m = -1;
  ctl->qnt_r = -1;
  ctl->qnt_rho = -1;
  ctl->qnt_ps = -1;
  ctl->qnt_pt = -1;
  ctl->qnt_z = -1;
  ctl->qnt_p = -1;
  ctl->qnt_t = -1;
  ctl->qnt_u = -1;
  ctl->qnt_v = -1;
  ctl->qnt_w = -1;
  ctl->qnt_h2o = -1;
  ctl->qnt_o3 = -1;
  ctl->qnt_theta = -1;
  ctl->qnt_vh = -1;
  ctl->qnt_vz = -1;
  ctl->qnt_pv = -1;
  ctl->qnt_tice = -1;
  ctl->qnt_tsts = -1;
  ctl->qnt_tnat = -1;
  ctl->qnt_stat = -1;

  /* Read quantities... */
  ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
  if (ctl->nq > NQ)
    ERRMSG("Too many quantities!");
  for (iq = 0; iq < ctl->nq; iq++) {

    /* Read quantity name and format... */
    scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
    scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
             ctl->qnt_format[iq]);

    /* Try to identify quantity... */
    if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
      ctl->qnt_ens = iq;
      sprintf(ctl->qnt_unit[iq], "-");
    } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
      ctl->qnt_m = iq;
      sprintf(ctl->qnt_unit[iq], "kg");
    } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
      ctl->qnt_r = iq;
      sprintf(ctl->qnt_unit[iq], "m");
    } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
      ctl->qnt_rho = iq;
      sprintf(ctl->qnt_unit[iq], "kg/m^3");
    } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
      ctl->qnt_ps = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
      ctl->qnt_pt = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
      ctl->qnt_z = iq;
      sprintf(ctl->qnt_unit[iq], "km");
    } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
      ctl->qnt_p = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
      ctl->qnt_t = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
      ctl->qnt_u = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
      ctl->qnt_v = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
      ctl->qnt_w = iq;
      sprintf(ctl->qnt_unit[iq], "hPa/s");
    } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
      ctl->qnt_h2o = iq;
      sprintf(ctl->qnt_unit[iq], "1");
    } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
      ctl->qnt_o3 = iq;
      sprintf(ctl->qnt_unit[iq], "1");
    } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
      ctl->qnt_theta = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
      ctl->qnt_vh = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
      ctl->qnt_vz = iq;
      sprintf(ctl->qnt_unit[iq], "m/s");
    } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
      ctl->qnt_pv = iq;
      sprintf(ctl->qnt_unit[iq], "PVU");
    } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
      ctl->qnt_tice = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
      ctl->qnt_tsts = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
      ctl->qnt_tnat = iq;
      sprintf(ctl->qnt_unit[iq], "K");
    } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
      ctl->qnt_stat = iq;
      sprintf(ctl->qnt_unit[iq], "-");
    } else
      scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
  }

  /* Time steps of simulation... */
  ctl->direction =
    (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
  if (ctl->direction != -1 && ctl->direction != 1)
    ERRMSG("Set DIRECTION to -1 or 1!");
  ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
  ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);

  /* Meteorological data... */
  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
  ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
  ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
  ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
  ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
  ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
  ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
  ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
  if (ctl->met_np > EP)
    ERRMSG("Too many levels!");
  for (ip = 0; ip < ctl->met_np; ip++)
    ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
  ctl->met_tropo
    = (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "MET_GEOPOT", -1, "-", ctl->met_geopot);
  scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
  ctl->met_dt_out =
    scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);

  /* Isosurface parameters... */
  ctl->isosurf
    = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

  /* Diffusion parameters... */
  ctl->turb_dx_trop
    = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
  ctl->turb_dx_strat
    = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
  ctl->turb_dz_trop
    = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
  ctl->turb_dz_strat
    = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
  ctl->turb_mesox =
    scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
  ctl->turb_mesoz =
    scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);

  /* Mass and life time... */
  ctl->molmass = scan_ctl(filename, argc, argv, "MOLMASS", -1, "1", NULL);
  ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
  ctl->tdec_strat =
    scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);

  /* PSC analysis... */
  ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
  ctl->psc_hno3 =
    scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);

  /* Output of atmospheric data... */
  scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
  scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
  ctl->atm_dt_out =
    scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
  ctl->atm_filter =
    (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
  ctl->atm_type =
    (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);

  /* Output of CSI data... */
  scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
  ctl->csi_dt_out =
    scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
  scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
  ctl->csi_obsmin =
    scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
  ctl->csi_modmin =
    scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
  ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
  ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
  ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
  ctl->csi_lon0 =
    scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
  ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
  ctl->csi_nx =
    (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
  ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
  ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
  ctl->csi_ny =
    (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);

  /* Output of ensemble data... */
  scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);

  /* Output of grid data... */
  scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
           ctl->grid_basename);
  scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
  ctl->grid_dt_out =
    scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
  ctl->grid_sparse =
    (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
  ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
  ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
  ctl->grid_nz =
    (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
  ctl->grid_lon0 =
    scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
  ctl->grid_lon1 =
    scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
  ctl->grid_nx =
    (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
  ctl->grid_lat0 =
    scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
  ctl->grid_lat1 =
    scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
  ctl->grid_ny =
    (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);

  /* Output of profile data... */
  scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
           ctl->prof_basename);
  scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
  ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
  ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
  ctl->prof_nz =
    (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
  ctl->prof_lon0 =
    scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
  ctl->prof_lon1 =
    scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
  ctl->prof_nx =
    (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
  ctl->prof_lat0 =
    scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
  ctl->prof_lat1 =
    scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
  ctl->prof_ny =
    (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);

  /* Output of station data... */
  scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
           ctl->stat_basename);
  ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
  ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
  ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
}
```

Here is the call graph for this function:



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

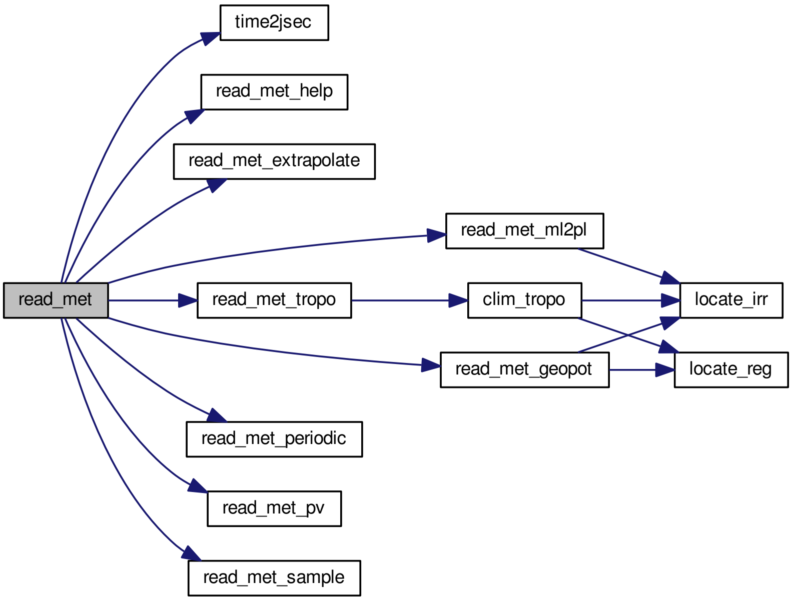
Read meteorological data file.

Definition at line 1355 of file libtrac.c.

```
               {

  char cmd[2 * LEN], levname[LEN], tstr[10];

  float help[EX * EY];

  int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;

  size_t np, nx, ny;

  /* Write info... */
  printf("Read meteorological data: %s\n", filename);

  /* Open netCDF file... */
  if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {

    /* Try to stage meteo file... */
    if (ctl->met_stage[0] != '-') {
      sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
              year, mon, day, hour, filename);
      if (system(cmd) != 0)
        ERRMSG("Error while staging meteo data!");
    }

    /* Try to open again... */
    if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
      return 0;
  }

  /* Get time from filename... */
  sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
  hour = atoi(tstr);
  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx < 2 || nx > EX)
    ERRMSG("Number of longitudes out of range!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");

  sprintf(levname, "lev");
  NC(nc_inq_dimid(ncid, levname, &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
  if (np == 1) {
    sprintf(levname, "lev_2");
    NC(nc_inq_dimid(ncid, levname, &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
  }
  if (np < 2 || np > EP)
    ERRMSG("Number of levels out of range!");

  /* Store dimensions... */
  met->np = (int) np;
  met->nx = (int) nx;
  met->ny = (int) ny;

  /* Get horizontal grid... */
  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));
  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

  /* Read meteorological data... */
  read_met_help(ncid, "t", "T", met, met->t, 1.0);
  read_met_help(ncid, "u", "U", met, met->u, 1.0);
  read_met_help(ncid, "v", "V", met, met->v, 1.0);
  read_met_help(ncid, "w", "W", met, met->w, 0.01f);
  read_met_help(ncid, "q", "Q", met, met->h2o, (float) (MA / 18.01528));
  read_met_help(ncid, "o3", "O3", met, met->o3, (float) (MA / 48.00));

  /* Meteo data on pressure levels... */
  if (ctl->met_np <= 0) {

    /* Read pressure levels from file... */
    NC(nc_inq_varid(ncid, levname, &varid));
    NC(nc_get_var_double(ncid, varid, met->p));
    for (ip = 0; ip < met->np; ip++)
      met->p[ip] /= 100.;

    /* Extrapolate data for lower boundary... */
    read_met_extrapolate(met);
  }

  /* Meteo data on model levels... */
  else {

    /* Read pressure data from file... */
    read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);

    /* Interpolate from model levels to pressure levels... */
    read_met_ml2pl(ctl, met, met->t);
    read_met_ml2pl(ctl, met, met->u);
    read_met_ml2pl(ctl, met, met->v);
    read_met_ml2pl(ctl, met, met->w);
    read_met_ml2pl(ctl, met, met->h2o);
    read_met_ml2pl(ctl, met, met->o3);

    /* Set pressure levels... */
    met->np = ctl->met_np;
    for (ip = 0; ip < met->np; ip++)
      met->p[ip] = ctl->met_p[ip];
  }

  /* Check ordering of pressure levels... */
  for (ip = 1; ip < met->np; ip++)
    if (met->p[ip - 1] < met->p[ip])
      ERRMSG("Pressure levels must be descending!");

  /* Read surface pressure... */
  if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
      || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
```

```
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
  } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
             || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
  } else
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->ps[ix][iy] = met->p[0];

  /* Create periodic boundary conditions... */
  read_met_periodic(met);

  /* Calculate geopotential heights... */
  read_met_geopot(ctl, met);

  /* Calculate potential vorticity... */
  read_met_pv(met);

  /* Calculate tropopause pressure... */
  read_met_tropo(ctl, met);

  /* Downsampling... */
  read_met_sample(ctl, met);

  /* Close file... */
  NC(nc_close(ncid));

  /* Return success... */
  return 1;
}
```

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.

```
               {

  int ip, ip0, ix, iy;

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      for (ip0 = met->np - 1; ip0 >= 0; ip0--)
        if (!gsl_finite(met->t[ix][iy][ip0])
            || !gsl_finite(met->u[ix][iy][ip0])
            || !gsl_finite(met->v[ix][iy][ip0])
            || !gsl_finite(met->w[ix][iy][ip0]))
          break;

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--) {
        met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
        met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
        met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
        met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
        met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
        met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
      }
    }
}
```

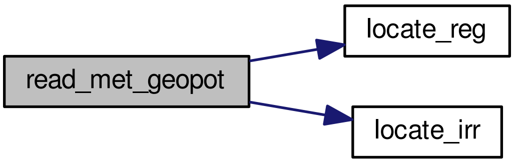
5.21.2.21 void read_met_geopot (ctl_t * *ctl*, met_t * *met*)

Calculate geopotential heights.

Definition at line 1551 of file libtrac.c.

```
               {

  static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];

  static int init, topo_nx = -1, topo_ny;

  FILE *in;

  char line[LEN];

  double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;

  float help[EX][EY];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;

  /* Initialize geopotential heights... */
#pragma omp parallel for default(shared) private(ix,iy,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++)
        met->z[ix][iy][ip] = GSL_NAN;

  /* Check filename... */
  if (ctl->met_geopot[0] == '-')
    return;

  /* Read surface geopotential... */
  if (!init) {
    init = 1;

    /* Write info... */
    printf("Read surface geopotential: %s\n", ctl->met_geopot);

    /* Open file... */
    if (!(in = fopen(ctl->met_geopot, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
        if (rlon != rlon_old) {
          if ((++topo_nx) >= EX)
            ERRMSG("Too many longitudes!");
          topo_ny = 0;
        }
        rlon_old = rlon;
        topo_lon[topo_nx] = rlon;
        topo_lat[topo_ny] = rlat;
        topo_z[topo_nx][topo_ny] = rz;
        if ((++topo_ny) >= EY)
          ERRMSG("Too many latitudes!");
      }
    if ((++topo_nx) >= EX)
      ERRMSG("Too many longitudes!");

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

    /* Check grid spacing... */
    if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
        || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
      printf("Warning: Grid spacing does not match!\n");
  }

  /* Apply hydrostatic equation to calculate geopotential heights... */
#pragma omp parallel for default(shared) private(ix,iy,lon,lat,tx,ty,z0,z1,ip0,ts,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Get surface height... */
      lon = met->lon[ix];
      if (lon < topo_lon[0])
        lon += 360;
      else if (lon > topo_lon[topo_nx - 1])
        lon -= 360;
      lat = met->lat[iy];
      tx = locate_reg(topo_lon, topo_nx, lon);
      ty = locate_reg(topo_lat, topo_ny, lat);
      z0 = LIN(topo_lon[tx], topo_z[tx][ty],
               topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
      z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
               topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
      z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);

      /* Find surface pressure level... */
      ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);

      /* Get surface temperature... */
      ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
               met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (z0 + RI / MA / G0 * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
                   * log(met->ps[ix][iy] / met->p[ip0 + 1]));
      for (ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0
                     * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
                     * log(met->p[ip - 1] / met->p[ip]));
    }

  /* Smooth fields... */
  for (ip = 0; ip < met->np; ip++) {

    /* Median filter... */
#pragma omp parallel for default(shared) private(ix,iy,n,ix2,ix3,iy2,data)
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {
        n = 0;
        for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
               iy2++)
            if (gsl_finite(met->z[ix3][iy2][ip])) {
              data[n] = met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0) {
          gsl_sort(data, 1, (size_t) n);
          help[ix][iy] = (float)
            gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
        } else
          help[ix][iy] = GSL_NAN;
      }

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

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.

```
                {

  float *help;

  int ip, ix, iy, varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      return;

  /* Allocate... */
  ALLOC(help, float, met->nx * met->ny * met->np);

  /* Read data... */
  NC(nc_get_var_float(ncid, varid, help));

  /* Copy and check data... */
#pragma omp parallel for default(shared) private(ix,iy,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
        if (fabsf(dest[ix][iy][ip]) < 1e14f)
          dest[ix][iy][ip] *= scl;
        else
          dest[ix][iy][ip] = GSL_NAN;
      }

  /* Free... */
  free(help);
}
```

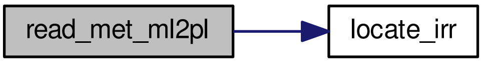
5.21.2.23 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 1736 of file libtrac.c.

```
                         {

  double aux[EP], p[EP], pt;

  int ip, ip2, ix, iy;

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Copy pressure profile... */
      for (ip = 0; ip < met->np; ip++)
        p[ip] = met->pl[ix][iy][ip];

      /* Interpolate... */
      for (ip = 0; ip < ctl->met_np; ip++) {
        pt = ctl->met_p[ip];
        if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
          pt = p[0];
        else if ((pt > p[met->np - 1] && p[1] > p[0])
                 || (pt < p[met->np - 1] && p[1] < p[0]))
          pt = p[met->np - 1];
        ip2 = locate_irr(p, met->np, pt);
        aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
                      p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
      }

      /* Copy data... */
      for (ip = 0; ip < ctl->met_np; ip++)
        var[ix][iy][ip] = (float) aux[ip];
    }
}
```

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.

```
               {

  int ip, iy;

  /* Check longitudes... */
  if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
             + met->lon[1] - met->lon[0] - 360) < 0.01))
    return;

  /* Increase longitude counter... */
  if ((++met->nx) > EX)
    ERRMSG("Cannot create periodic boundary conditions!");

  /* Set longitude... */
  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];

  /* Loop over latitudes and pressure levels... */
#pragma omp parallel for default(shared) private(iy,ip)
  for (iy = 0; iy < met->ny; iy++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->pt[met->nx - 1][iy] = met->pt[0][iy];
    for (ip = 0; ip < met->np; ip++) {
      met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
      met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
      met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
      met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
      met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
      met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
      met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
    }
  }
}
```

5.21.2.25 void read_met_pv (met_t * *met*)

Calculate potential vorticity.

Definition at line 1812 of file libtrac.c.

```
               {

  double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
    dtdp, dudp, dvdp, latr, vort, pows[EP];

  int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;

  /* Set powers... */
  for (ip = 0; ip < met->np; ip++)
    pows[ip] = pow(1000. / met->p[ip], 0.286);

  /* Loop over grid points... */
#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)
  for (ix = 0; ix < met->nx; ix++) {

    /* Set indices... */
    ix0 = GSL_MAX(ix - 1, 0);
    ix1 = GSL_MIN(ix + 1, met->nx - 1);

    /* Loop over grid points... */
    for (iy = 0; iy < met->ny; iy++) {

      /* Set indices... */
      iy0 = GSL_MAX(iy - 1, 0);
      iy1 = GSL_MIN(iy + 1, met->ny - 1);

      /* Set auxiliary variables... */
      latr = GSL_MIN(GSL_MAX(met->lat[iy], -89.), 89.);
      dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
      dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
      c0 = cos(met->lat[iy0] / 180. * M_PI);
      c1 = cos(met->lat[iy1] / 180. * M_PI);
      cr = cos(latr / 180. * M_PI);
      vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);

      /* Loop over grid points... */
      for (ip = 0; ip < met->np; ip++) {

        /* Get gradients in longitude... */
        dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
        dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;

        /* Get gradients in latitude... */
        dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
        dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;

        /* Set indices... */
        ip0 = GSL_MAX(ip - 1, 0);
        ip1 = GSL_MIN(ip + 1, met->np - 1);

        /* Get gradients in pressure... */
        dp0 = 100. * (met->p[ip] - met->p[ip0]);
        dp1 = 100. * (met->p[ip1] - met->p[ip]);
        if (ip != ip0 && ip != ip1) {
          denom = dp0 * dp1 * (dp0 + dp1);
          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
                  - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
            / denom;
          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
                  - dp1 * dp1 * met->u[ix][iy][ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
            / denom;
          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
                  - dp1 * dp1 * met->v[ix][iy][ip0]
                  + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
            / denom;
        } else {
          denom = dp0 + dp1;
          dtdp =
            (met->t[ix][iy][ip1] * pows[ip1] -
             met->t[ix][iy][ip0] * pows[ip0]) / denom;
          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
        }

        /* Calculate PV... */
        met->pv[ix][iy][ip] = (float)
          (1e6 * G0 *
           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
      }
    }
  }
}
```

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.

```
               {

  met_t *help;

  float w, wsum;

  int ip, ip2, ix, ix2, ix3, iy, iy2;

  /* Check parameters... */
  if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
      && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
    return;

  /* Allocate... */
  ALLOC(help, met_t, 1);

  /* Copy data... */
  help->nx = met->nx;
  help->ny = met->ny;
  help->np = met->np;
  memcpy(help->lon, met->lon, sizeof(met->lon));
  memcpy(help->lat, met->lat, sizeof(met->lat));
  memcpy(help->p, met->p, sizeof(met->p));

  /* Smoothing... */
  for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
    for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
      for (ip = 0; ip < met->np; ip += ctl->met_dp) {
        help->ps[ix][iy] = 0;
        help->pt[ix][iy] = 0;
        help->z[ix][iy][ip] = 0;
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
        help->pv[ix][iy][ip] = 0;
        help->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        wsum = 0;
        for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
            for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
              w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
                * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
                * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->pt[ix][iy] += w * met->pt[ix3][iy2];
              help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
              help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
              help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
              help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
              help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
              help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
              help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->pt[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->z[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->pv[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Downsampling... */
  met->nx = 0;
  for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
      met->lat[met->ny] = help->lat[iy];
      met->ps[met->nx][met->ny] = help->ps[ix][iy];
      met->pt[met->nx][met->ny] = help->pt[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl->met_dp) {
        met->p[met->np] = help->p[ip];
        met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
        met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
        met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
        met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
        met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
        met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
        met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
        met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

  /* Free... */
  free(help);
}
```

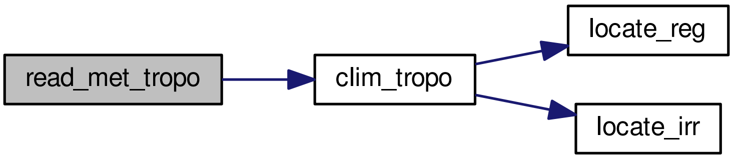
5.21.2.27 void read_met_tropo (ctl_t * *ctl*, met_t * *met*)

Calculate tropopause pressure.

Definition at line 2015 of file libtrac.c.

```
             {

gsl_interp_accel *acc;

gsl_spline *spline;

double p2[400], pv[400], pv2[400], t[400], t2[400], th[400], th2[400],
  z[400], z2[400];

int found, ix, iy, iz, iz2;

/* Allocate... */
acc = gsl_interp_accel_alloc();
spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) met->np);

/* Get altitude and pressure profiles... */
for (iz = 0; iz < met->np; iz++)
  z[iz] = Z(met->p[iz]);
for (iz = 0; iz <= 170; iz++) {
  z2[iz] = 4.5 + 0.1 * iz;
  p2[iz] = P(z2[iz]);
}

/* Do not calculate tropopause... */
if (ctl->met_tropo == 0)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      met->pt[ix][iy] = GSL_NAN;

/* Use tropopause climatology... */
else if (ctl->met_tropo == 1)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      met->pt[ix][iy] = clim_tropo(met->time, met->lat[iy]);

/* Use cold point... */
else if (ctl->met_tropo == 2) {

  /* Loop over grid points... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Interpolate temperature profile... */
      for (iz = 0; iz < met->np; iz++)
        t[iz] = met->t[ix][iy][iz];
      gsl_spline_init(spline, z, t, (size_t) met->np);
      for (iz = 0; iz <= 170; iz++)
        t2[iz] = gsl_spline_eval(spline, z2[iz], acc);

      /* Find minimum... */
      iz = (int) gsl_stats_min_index(t2, 1, 171);
      if (iz <= 0 || iz >= 170)
        met->pt[ix][iy] = GSL_NAN;
      else
        met->pt[ix][iy] = p2[iz];
    }
}

/* Use WMO definition... */
else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {

  /* Loop over grid points... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Interpolate temperature profile... */
      for (iz = 0; iz < met->np; iz++)
        t[iz] = met->t[ix][iy][iz];
      gsl_spline_init(spline, z, t, (size_t) met->np);
      for (iz = 0; iz <= 160; iz++)
        t2[iz] = gsl_spline_eval(spline, z2[iz], acc);

      /* Find 1st tropopause... */
      met->pt[ix][iy] = GSL_NAN;
      for (iz = 0; iz <= 140; iz++) {
        found = 1;
        for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
          if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
              / log(p2[iz2] / p2[iz]) > 2.0) {
            found = 0;
            break;
          }
        if (found) {
          if (iz > 0 && iz < 140)
            met->pt[ix][iy] = p2[iz];
          break;
        }
        }

        /* Find 2nd tropopause... */
        if (ctl->met_tropo == 4) {
          met->pt[ix][iy] = GSL_NAN;
          for (; iz <= 140; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
              if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                  / log(p2[iz2] / p2[iz]) < 3.0) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 140; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
              if (1000. * G0 / RA * log(t2[iz2] / t2[iz])
                  / log(p2[iz2] / p2[iz]) > 2.0) {
                found = 0;
                break;
              }
            if (found) {
              if (iz > 0 && iz < 140)
                met->pt[ix][iy] = p2[iz];
              break;
            }
          }
        }
      }
  }

  /* Use dynamical tropopause... */
  else if (ctl->met_tropo == 5) {

    /* Loop over grid points... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++) {

        /* Interpolate potential vorticity profile... */
        for (iz = 0; iz < met->np; iz++)
          pv[iz] = met->pv[ix][iy][iz];
        gsl_spline_init(spline, z, pv, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          pv2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Interpolate potential temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
        gsl_spline_init(spline, z, th, (size_t) met->np);
        for (iz = 0; iz <= 160; iz++)
          th2[iz] = gsl_spline_eval(spline, z2[iz], acc);

        /* Find dynamical tropopause 3.5 PVU + 380 K */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 160; iz++)
          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
            if (iz > 0 && iz < 160)
              met->pt[ix][iy] = p2[iz];
            break;
          }
      }
  }

  else
    ERRMSG("Cannot calculate tropopause!");

  /* Free... */
  gsl_spline_free(spline);
  gsl_interp_accel_free(acc);
}
```

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.

```
        {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    msg[2 * LEN], rvarname[LEN], rval[LEN];

  int contain = 0, i;

  /* Open file... */
  if (filename[strlen(filename) - 1] != '-')
    if (!(in = fopen(filename, "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
        }
  for (i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else {
      sprintf(msg, "Missing variable %s!\n", fullname1);
      ERRMSG(msg);
    }
  }

  /* Write info... */
  printf("%s = %s\n", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}
```

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.

```
                {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

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.

```
            {

  static double starttime[NTIMER], runtime[NTIMER];

  /* Check id... */
  if (id < 0 || id >= NTIMER)
    ERRMSG("Too many timers!");

  /* Start timer... */
  if (mode == 1) {
    if (starttime[id] <= 0)
      starttime[id] = omp_get_wtime();
    else
      ERRMSG("Timer already started!");
  }

  /* Stop timer... */
  else if (mode == 2) {
    if (starttime[id] > 0) {
      runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
      starttime[id] = -1;
    }
  }

  /* Print timer... */
  else if (mode == 3) {
    printf("%s = %.3f s\n", name, runtime[id]);
    runtime[id] = 0;
  }
}
```

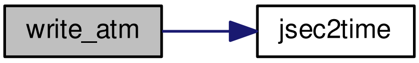
5.21.2.31 void write_atm (const char * *filename,* ctl_t * *ctl,* atm_t * *atm,* double *t*)

Write atmospheric data.

Definition at line 2319 of file libtrac.c.

```
            {

  FILE *in, *out;

  char line[LEN];

  double r, t0, t1;

  int ip, iq, year, mon, day, hour, min, sec;

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Write info... */
  printf("Write atmospheric data: %s\n", filename);

  /* Write ASCII data... */
  if (ctl->atm_type == 0) {

    /* Check if gnuplot output is requested... */
    if (ctl->atm_gpfile[0] != '-') {

      /* Create gnuplot pipe... */
      if (!(out = popen("gnuplot", "w")))
        ERRMSG("Cannot create pipe to gnuplot!");

      /* Set plot filename... */
      fprintf(out, "set out \"%s.png\"\n", filename);

      /* Set time string... */
      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
      fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
              year, mon, day, hour, min);

      /* Dump gnuplot file to pipe... */
      if (!(in = fopen(ctl->atm_gpfile, "r")))
        ERRMSG("Cannot open file!");
      while (fgets(line, LEN, in))
        fprintf(out, "%s", line);
      fclose(in);
    }

    else {

      /* Create file... */
      if (!(out = fopen(filename, "w")))
        ERRMSG("Cannot create file!");
    }

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
              ctl->qnt_unit[iq]);
    fprintf(out, "\n");

    /* Write data... */
    for (ip = 0; ip < atm->np; ip++) {

      /* Check time... */
      if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
        continue;

      /* Write output... */
      fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
              atm->lon[ip], atm->lat[ip]);
      for (iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
      }
      fprintf(out, "\n");
    }

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

  /* Write binary data... */
  else if (ctl->atm_type == 1) {

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write data... */
    FWRITE(&atm->np, int,
           1,
           out);
    FWRITE(atm->time, double,
             (size_t) atm->np,
           out);
    FWRITE(atm->p, double,
             (size_t) atm->np,
           out);
    FWRITE(atm->lon, double,
             (size_t) atm->np,
           out);
    FWRITE(atm->lat, double,
             (size_t) atm->np,
           out);
    for (iq = 0; iq < ctl->nq; iq++)
      FWRITE(atm->q[iq], double,
               (size_t) atm->np,
             out);

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

  /* Error... */
  else
    ERRMSG("Atmospheric data type not supported!");
}
```

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.

```
            {

  static FILE *in, *out;

  static char line[LEN];

  static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
    rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;

  static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantity index for mass... */
    if (ctl->qnt_m < 0)
      ERRMSG("Need quantity mass!");

    /* Open observation data file... */
    printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
    if (!(in = fopen(ctl->csi_obsfile, "r")))
      ERRMSG("Cannot open file!");

    /* Create new file... */
    printf("Write CSI data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = number of hits (cx)\n"
            "# $3 = number of misses (cy)\n"
            "# $4 = number of false alarms (cz)\n"
            "# $5 = number of observations (cx + cy)\n"
            "# $6 = number of forecasts (cx + cz)\n"
            "# $7 = bias (forecasts/observations) [%%]\n"
            "# $8 = probability of detection (POD) [%%]\n"
            "# $9 = false alarm rate (FAR) [%%]\n"
            "# $10 = critical success index (CSI) [%%]\n\n");
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Initialize grid cells... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->csi_nx; ix++)
    for (iy = 0; iy < ctl->csi_ny; iy++)
      for (iz = 0; iz < ctl->csi_nz; iz++)
        modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;

  /* Read observation data... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
        5)
      continue;

    /* Check time... */
    if (rt < t0)
      continue;
    if (rt > t1)
      break;

    /* Calculate indices... */
    ix = (int) ((rlon - ctl->csi_lon0)
                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
    iy = (int) ((rlat - ctl->csi_lat0)
                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
    iz = (int) ((rz - ctl->csi_z0)
                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->csi_nx ||
        iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
      continue;

    /* Get mean observation index... */
    obsmean[ix][iy][iz] += robs;
    obscount[ix][iy][iz]++;
  }

  /* Analyze model data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Get indices... */
    ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
    iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
    iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->csi_nx ||
        iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
      continue;

    /* Get total mass in grid cell... */
    modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  }

  /* Analyze all grid cells... */
#pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
  for (ix = 0; ix < ctl->csi_nx; ix++)
    for (iy = 0; iy < ctl->csi_ny; iy++)
      for (iz = 0; iz < ctl->csi_nz; iz++) {

        /* Calculate mean observation index... */
        if (obscount[ix][iy][iz] > 0)
          obsmean[ix][iy][iz] /= obscount[ix][iy][iz];

        /* Calculate column density... */
        if (modmean[ix][iy][iz] > 0) {
          dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
          dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
          lat = ctl->csi_lat0 + dlat * (iy + 0.5);
          area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
            * cos(lat * M_PI / 180.);
          modmean[ix][iy][iz] /= (1e6 * area);
        }

        /* Calculate CSI... */
        if (obscount[ix][iy][iz] > 0) {
          if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
              modmean[ix][iy][iz] >= ctl->csi_modmin)
            cx++;
          else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
                   modmean[ix][iy][iz] < ctl->csi_modmin)
            cy++;
          else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
                   modmean[ix][iy][iz] >= ctl->csi_modmin)
            cz++;
        }
      }

  /* Write output... */
  if (fmod(t, ctl->csi_dt_out) == 0) {

    /* Write... */
    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
            t, cx, cy, cz, cx + cy, cx + cz,
            (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
            (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
            (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
            (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);

    /* Set counters to zero... */
    cx = cy = cz = 0;
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

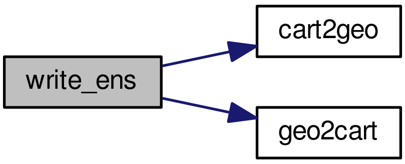
5.21.2.33 void write_ens (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

Write ensemble data.

Definition at line 2613 of file libtrac.c.

```
            {

  static FILE *out;

  static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
    t0, t1, x[NENS][3], xm[3];

  static int ip, iq;

  static size_t i, n;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantities... */
    if (ctl->qnt_ens < 0)
      ERRMSG("Missing ensemble IDs!");

    /* Create new file... */
    printf("Write ensemble data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    for (iq = 0; iq < ctl->nq; iq++)
      fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
              ctl->qnt_name[iq], ctl->qnt_unit[iq]);
    fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Init... */
  ens = GSL_NAN;
  n = 0;

  /* Loop over air parcels... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Check ensemble id... */
    if (atm->q[ctl->qnt_ens][ip] != ens) {

      /* Write results... */
      if (n > 0) {

        /* Get mean position... */
        xm[0] = xm[1] = xm[2] = 0;
        for (i = 0; i < n; i++) {
          xm[0] += x[i][0] / (double) n;
          xm[1] += x[i][1] / (double) n;
          xm[2] += x[i][2] / (double) n;
        }
        cart2geo(xm, &dummy, &lon, &lat);
        fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
                lat);

        /* Get quantity statistics... */
        for (iq = 0; iq < ctl->nq; iq++) {
          fprintf(out, " ");
          fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
        }
        for (iq = 0; iq < ctl->nq; iq++) {
          fprintf(out, " ");
          fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
        }
        fprintf(out, " %lu\n", n);
      }

      /* Init new ensemble... */
      ens = atm->q[ctl->qnt_ens][ip];
      n = 0;
    }

    /* Save data... */
    p[n] = atm->p[ip];
    geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
    for (iq = 0; iq < ctl->nq; iq++)
      q[iq][n] = atm->q[iq][ip];
    if ((++n) >= NENS)
      ERRMSG("Too many data points!");
  }

  /* Write results... */
  if (n > 0) {

    /* Get mean position... */
    xm[0] = xm[1] = xm[2] = 0;
    for (i = 0; i < n; i++) {
      xm[0] += x[i][0] / (double) n;
      xm[1] += x[i][1] / (double) n;
      xm[2] += x[i][2] / (double) n;
    }
    cart2geo(xm, &dummy, &lon, &lat);
    fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);

    /* Get quantity statistics... */
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
    }
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
    }
    fprintf(out, " %lu\n", n);
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

Here is the call graph for this function:



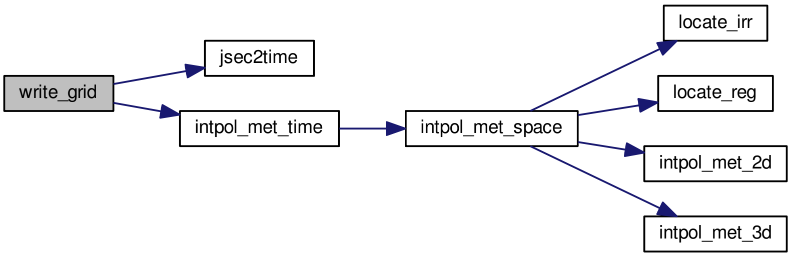
5.21.2.34 void write_grid (const char ∗ *filename,* ctl_t ∗ *ctl,* met_t ∗ *met0,* met_t ∗ *met1,* atm_t ∗ *atm,* double *t*)

Write gridded data.

Definition at line 2744 of file libtrac.c.

```
            {

  FILE *in, *out;

  char line[LEN];

  static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
    area, rho_air, press, temp, cd, vmr, t0, t1, r;

  static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec;

  /* Check dimensions... */
  if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
    ERRMSG("Grid dimensions too large!");

  /* Set time interval for output... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Set grid box size... */
  dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
  dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
  dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;

  /* Initialize grid... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->grid_nx; ix++)
    for (iy = 0; iy < ctl->grid_ny; iy++)
      for (iz = 0; iz < ctl->grid_nz; iz++) {
        mass[ix][iy][iz] = 0;
        np[ix][iy][iz] = 0;
      }

  /* Average data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++)
    if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {

      /* Get index... */
      ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
      iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
      iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);

      /* Check indices... */
      if (ix < 0 || ix >= ctl->grid_nx ||
          iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
        continue;

      /* Add mass... */
      if (ctl->qnt_m >= 0)
        mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
      np[ix][iy][iz]++;
    }

  /* Check if gnuplot output is requested... */
  if (ctl->grid_gpfile[0] != '-') {

    /* Write info... */
    printf("Plot grid data: %s.png\n", filename);

    /* Create gnuplot pipe... */
    if (!(out = popen("gnuplot", "w")))
      ERRMSG("Cannot create pipe to gnuplot!");

    /* Set plot filename... */
    fprintf(out, "set out \"%s.png\"\n", filename);

    /* Set time string... */
    jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
    fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
            year, mon, day, hour, min);

    /* Dump gnuplot file to pipe... */
    if (!(in = fopen(ctl->grid_gpfile, "r")))
      ERRMSG("Cannot open file!");
    while (fgets(line, LEN, in))
      fprintf(out, "%s", line);
    fclose(in);
  }

  else {

    /* Write info... */
    printf("Write grid data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");
  }

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = surface area [km^2]\n"
          "# $6 = layer width [km]\n"
          "# $7 = number of particles [1]\n"
          "# $8 = column density [kg/m^2]\n"
          "# $9 = volume mixing ratio [1]\n\n");

  /* Write data... */
  for (ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
      fprintf(out, "\n");
    for (iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
        fprintf(out, "\n");
      for (iz = 0; iz < ctl->grid_nz; iz++)
        if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {

          /* Set coordinates... */
          z = ctl->grid_z0 + dz * (iz + 0.5);
          lon = ctl->grid_lon0 + dlon * (ix + 0.5);
          lat = ctl->grid_lat0 + dlat * (iy + 0.5);

          /* Get pressure and temperature... */
          press = P(z);
          intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                          NULL, &temp, NULL, NULL, NULL, NULL, NULL, NULL);

          /* Calculate surface area... */
          area = dlat * dlon * SQR(RE * M_PI / 180.)
            * cos(lat * M_PI / 180.);

          /* Calculate column density... */
          cd = mass[ix][iy][iz] / (1e6 * area);

          /* Calculate volume mixing ratio... */
          rho_air = 100. * press / (RA * temp);
          vmr = MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * 1e6 * area * 1e3 * dz);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
                  t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
        }
    }
  }

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

Here is the call graph for this function:



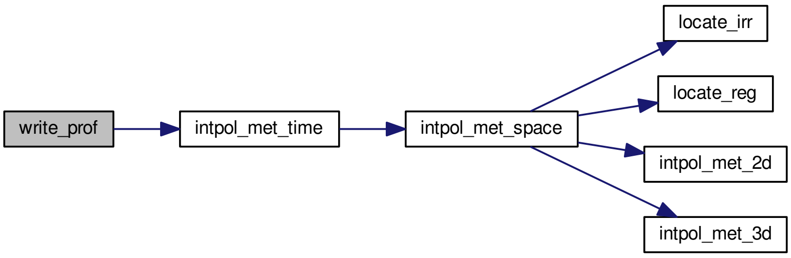
5.21.2.35 void write_prof (const char * *filename,* ctl_t * *ctl,* met_t * *met0,* met_t * *met1,* atm_t * *atm,* double *t*)

Write profile data.

Definition at line 2897 of file libtrac.c.

```
            {

  static FILE *in, *out;

  static char line[LEN];

  static double mass[GX][GY][GZ], obsmean[GX][GY], obsmean2[GX][GY], rt, rz,
    rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp,
    rho_air, vmr, h2o, o3;

  static int obscount[GX][GY], ip, ix, iy, iz, okay;

  /* Init... */
  if (t == ctl->t_start) {

    /* Check quantity index for mass... */
    if (ctl->qnt_m < 0)
      ERRMSG("Need quantity mass!");

    /* Check dimensions... */
    if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
      ERRMSG("Grid dimensions too large!");

    /* Open observation data file... */
    printf("Read profile observation data: %s\n", ctl->prof_obsfile);
    if (!(in = fopen(ctl->prof_obsfile, "r")))
      ERRMSG("Cannot open file!");

    /* Create new output file... */
    printf("Write profile data: %s\n", filename);
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n"
            "# $4 = latitude [deg]\n"
            "# $5 = pressure [hPa]\n"
            "# $6 = temperature [K]\n"
            "# $7 = volume mixing ratio [1]\n"
            "# $8 = H2O volume mixing ratio [1]\n"
            "# $9 = O3 volume mixing ratio [1]\n"
            "# $10 = observed BT index (mean) [K]\n"
            "# $11 = observed BT index (sigma) [K]\n");

    /* Set grid box size... */
    dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
    dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
    dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
  }

  /* Set time interval... */
  t0 = t - 0.5 * ctl->dt_mod;
  t1 = t + 0.5 * ctl->dt_mod;

  /* Initialize... */
#pragma omp parallel for default(shared) private(ix,iy,iz)
  for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++) {
      obsmean[ix][iy] = 0;
      obsmean2[ix][iy] = 0;
      obscount[ix][iy] = 0;
      for (iz = 0; iz < ctl->prof_nz; iz++)
        mass[ix][iy][iz] = 0;
    }

  /* Read observation data... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
        5)
      continue;

    /* Check time... */
    if (rt < t0)
      continue;
    if (rt > t1)
      break;

    /* Calculate indices... */
    ix = (int) ((rlon - ctl->prof_lon0) / dlon);
    iy = (int) ((rlat - ctl->prof_lat0) / dlat);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
      continue;

    /* Get mean observation index... */
    obsmean[ix][iy] += robs;
    obsmean2[ix][iy] += SQR(robs);
    obscount[ix][iy]++;
  }

  /* Analyze model data... */
#pragma omp parallel for default(shared) private(ip,ix,iy,iz)
  for (ip = 0; ip < atm->np; ip++) {

    /* Check time... */
    if (atm->time[ip] < t0 || atm->time[ip] > t1)
      continue;

    /* Get indices... */
    ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
    iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
    iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);

    /* Check indices... */
    if (ix < 0 || ix >= ctl->prof_nx ||
        iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
      continue;

    /* Get total mass in grid cell... */
    mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  }

  /* Extract profiles... */
  for (ix = 0; ix < ctl->prof_nx; ix++)
    for (iy = 0; iy < ctl->prof_ny; iy++)
      if (obscount[ix][iy] > 0) {

        /* Check profile... */
        okay = 0;
        for (iz = 0; iz < ctl->prof_nz; iz++)
          if (mass[ix][iy][iz] > 0) {
            okay = 1;
            break;
          }
        if (!okay)
          continue;

        /* Write output... */
        fprintf(out, "\n");

        /* Loop over altitudes... */
        for (iz = 0; iz < ctl->prof_nz; iz++) {

          /* Set coordinates... */
          z = ctl->prof_z0 + dz * (iz + 0.5);
          lon = ctl->prof_lon0 + dlon * (ix + 0.5);
          lat = ctl->prof_lat0 + dlat * (iy + 0.5);

          /* Get pressure and temperature... */
          press = P(z);
          intpol_met_time(met0, met1, t, press, lon, lat, NULL, NULL,
                          NULL, &temp, NULL, NULL, NULL, NULL, &h2o, &o3);

          /* Calculate surface area... */
          area = dlat * dlon * SQR(M_PI * RE / 180.)
            * cos(lat * M_PI / 180.);

          /* Calculate volume mixing ratio... */
          rho_air = 100. * press / (RA * temp);
          vmr = MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * area * dz * 1e9);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
                  t, z, lon, lat, press, temp, vmr, h2o, o3,
                  obsmean[ix][iy] / obscount[ix][iy],
                  sqrt(obsmean2[ix][iy] / obscount[ix][iy]
                       - SQR(obsmean[ix][iy] / obscount[ix][iy])));
        }
      }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

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*)

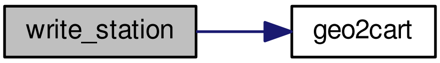
Write station data.

Definition at line 3077 of file libtrac.c.

```
          {

static FILE *out;

static double rmax2, t0, t1, x0[3], x1[3];

static int ip, iq;

/* Init... */
if (t == ctl->t_start) {

  /* Write info... */
  printf("Write station data: %s\n", filename);

  /* Create new file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
  for (iq = 0; iq < ctl->nq; iq++)
    fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
            ctl->qnt_name[iq], ctl->qnt_unit[iq]);
  fprintf(out, "\n");

  /* Set geolocation and search radius... */
  geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
  rmax2 = SQR(ctl->stat_r);
}

/* Set time interval for output... */
t0 = t - 0.5 * ctl->dt_mod;
t1 = t + 0.5 * ctl->dt_mod;

/* Loop over air parcels... */
for (ip = 0; ip < atm->np; ip++) {

  /* Check time... */
  if (atm->time[ip] < t0 || atm->time[ip] > t1)
    continue;

  /* Check station flag... */
  if (ctl->qnt_stat >= 0)
    if (atm->q[ctl->qnt_stat][ip])
      continue;

  /* Get Cartesian coordinates... */
```

```
    geo2cart(0, atm->lon[ip], atm->lat[ip], x1);

    /* Check horizontal distance... */
    if (DIST2(x0, x1) > rmax2)
      continue;

    /* Set station flag... */
    if (ctl->qnt_stat >= 0)
      atm->q[ctl->qnt_stat][ip] = 1;

    /* Write data... */
    fprintf(out, "%.2f %g %g %g",
            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
    for (iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

  /* Close file... */
  if (t == ctl->t_stop)
    fclose(out);
}
```

Here is the call graph for this function:



5.22 libtrac.h

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include <ctype.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_sort.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_statistics.h>
#include <math.h>
#include <netcdf.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

/* ------------------------------------------------------------
```

```
   Constants...
   ------------------------------------------------------------ */

#define G0 9.80665

#define H0 7.0

#define KB 1.3806504e-23

#define MA 28.9644

#define P0 1013.25

#define RA 287.058

#define RI 8.3144598

#define RE 6367.421

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define LEN 5000

#define NP 10000000

#define NQ 12

#define EP 112

#define EX 1201

#define EY 601

#define GX 720

#define GY 360

#define GZ 100

#define NENS 2000

#define NTHREADS 512

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

#define ALLOC(ptr, type, n)                                      \
  if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)      \
    ERRMSG("Out of memory!");

#define DEG2DX(dlon, lat)                                               \
  ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))

#define DEG2DY(dlat)                                   \
  ((dlat) * M_PI * RE / 180.)

#define DP2DZ(dp, p)                                   \
  (- (dp) * H0 / (p))

#define DX2DEG(dx, lat)                                                 \
  (((lat) < -89.999 || (lat) > 89.999) ? 0                              \
   : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))

#define DY2DEG(dy)                                     \
  ((dy) * 180. / (M_PI * RE))

#define DZ2DP(dz, p)                                   \
  (-(dz) * (p) / H0)

#define DIST(a, b) sqrt(DIST2(a, b))

#define DIST2(a, b)                                                             \
  ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))

#define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])

#define ERRMSG(msg) {                                                           \
    printf("\nError (%s, %s, l%d): %s\n\n",                                 \
           __FILE__, __func__, __LINE__, msg);                            \
    exit(EXIT_FAILURE);                                                       \
  }

#define FREAD(ptr, type, size, out) {                                         \
    if(fread(ptr, sizeof(type), size, out)!=size)                           \
      ERRMSG("Error while reading!");                                   \
  }

#define FWRITE(ptr, type, size, out) {                                  \
    if(fwrite(ptr, sizeof(type), size, out)!=size)                      \
      ERRMSG("Error while writing!");                                   \
  }

#define LIN(x0, y0, x1, y1, x)                  \
  ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))

#define NC(cmd) {                                    \
    if((cmd)!=NC_NOERR)                              \
      ERRMSG(nc_strerror(cmd));                      \
  }

#define NORM(a) sqrt(DOTP(a, a))

#define PRINT(format, var)                                              \
  printf("Print (%s, %s, l%d): %s= "format"\n",                         \
         __FILE__, __func__, __LINE__, #var, var);

#define P(z) (P0*exp(-(z)/H0))

#define SQR(x) ((x)*(x))

#define THETA(p, t) ((t)*pow(1000./(p), 0.286))

#define TOK(line, tok, format, var) {                                   \
    if(((tok)=strtok((line), " \t"))) {                                 \
      if(sscanf(tok, format, &(var))!=1) continue;                      \
    } else ERRMSG("Error while reading!");                              \
  }

#define Z(p) (H0*log(P0/(p)))

/* ------------------------------------------------------------
   Timers...
   ------------------------------------------------------------ */

#define START_TIMER(id) timer(#id, id, 1)

#define STOP_TIMER(id) timer(#id, id, 2)

#define PRINT_TIMER(id) timer(#id, id, 3)

#define NTIMER 12

#define TIMER_TOTAL 0

#define TIMER_INIT 1

#define TIMER_INPUT 2

#define TIMER_OUTPUT 3

#define TIMER_ADVECT 4

#define TIMER_DECAY 5

#define TIMER_DIFFMESO 6

#define TIMER_DIFFTURB 7

#define TIMER_ISOSURF 8

#define TIMER_METEO 9

#define TIMER_POSITION 10

#define TIMER_SEDI 11

/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */

typedef struct {

  int nq;

  char qnt_name[NQ][LEN];

  char qnt_unit[NQ][LEN];

  char qnt_format[NQ][LEN];

  int qnt_ens;

  int qnt_m;

  int qnt_rho;

  int qnt_r;

  int qnt_ps;

  int qnt_pt;

  int qnt_z;

  int qnt_p;

  int qnt_t;

  int qnt_u;

  int qnt_v;

  int qnt_w;

  int qnt_h2o;

  int qnt_o3;

  int qnt_theta;

  int qnt_vh;

  int qnt_vz;

  int qnt_pv;

  int qnt_tice;

  int qnt_tsts;

  int qnt_tnat;

  int qnt_stat;

  int direction;

  double t_start;

  double t_stop;

  double dt_mod;

  double dt_met;

  int met_dx;

  int met_dy;

  int met_dp;

  int met_sx;

  int met_sy;

  int met_sp;

  int met_np;

  double met_p[EP];

  int met_tropo;

  char met_geopot[LEN];

  double met_dt_out;

  char met_stage[LEN];

  int isosurf;

  char balloon[LEN];

  double turb_dx_trop;

  double turb_dx_strat;

  double turb_dz_trop;

  double turb_dz_strat;

  double turb_mesox;

  double turb_mesoz;

  double molmass;

  double tdec_trop;

  double tdec_strat;

  double psc_h2o;

  double psc_hno3;

  char atm_basename[LEN];

  char atm_gpfile[LEN];

  double atm_dt_out;

  int atm_filter;

  int atm_type;

  char csi_basename[LEN];

  double csi_dt_out;

  char csi_obsfile[LEN];

  double csi_obsmin;

  double csi_modmin;

  int csi_nz;

  double csi_z0;

  double csi_z1;

  int csi_nx;

  double csi_lon0;

  double csi_lon1;

  int csi_ny;

  double csi_lat0;

  double csi_lat1;

  char grid_basename[LEN];

  char grid_gpfile[LEN];

  double grid_dt_out;

  int grid_sparse;

  int grid_nz;

  double grid_z0;

  double grid_z1;

  int grid_nx;

  double grid_lon0;

  double grid_lon1;

  int grid_ny;

  double grid_lat0;

  double grid_lat1;

  char prof_basename[LEN];

  char prof_obsfile[LEN];

  int prof_nz;

  double prof_z0;

  double prof_z1;

  int prof_nx;

  double prof_lon0;

  double prof_lon1;

  int prof_ny;

  double prof_lat0;

  double prof_lat1;

  char ens_basename[LEN];

  char stat_basename[LEN];

  double stat_lon;

  double stat_lat;

  double stat_r;

} ctl_t;

typedef struct {

  int np;

  double time[NP];

  double p[NP];

  double lon[NP];

  double lat[NP];

  double q[NQ][NP];

  float up[NP];

  float vp[NP];

  float wp[NP];

  double cache_time[EX][EY][EP];

  float cache_usig[EX][EY][EP];

  float cache_vsig[EX][EY][EP];

  float cache_wsig[EX][EY][EP];

} atm_t;

typedef struct {

  double time;

  int nx;

  int ny;

  int np;

  double lon[EX];

  double lat[EY];

  double p[EP];

  double ps[EX][EY];

  double pt[EX][EY];

  float z[EX][EY][EP];

  float t[EX][EY][EP];

  float u[EX][EY][EP];

  float v[EX][EY][EP];

  float w[EX][EY][EP];

  float pv[EX][EY][EP];

  float h2o[EX][EY][EP];

  float o3[EX][EY][EP];

  float pl[EX][EY][EP];

} met_t;

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

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

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

double clim_tropo(
  double t,
  double lat);

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

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

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

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

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

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

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

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

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);

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);

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

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

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

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

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

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

void read_met_extrapolate(
  met_t * met);

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

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

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

```

```
void read_met_periodic(
  met_t * met);

void read_met_pv(
  met_t * met);

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

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

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

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

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

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

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

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

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

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

void write_station(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm,
  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.

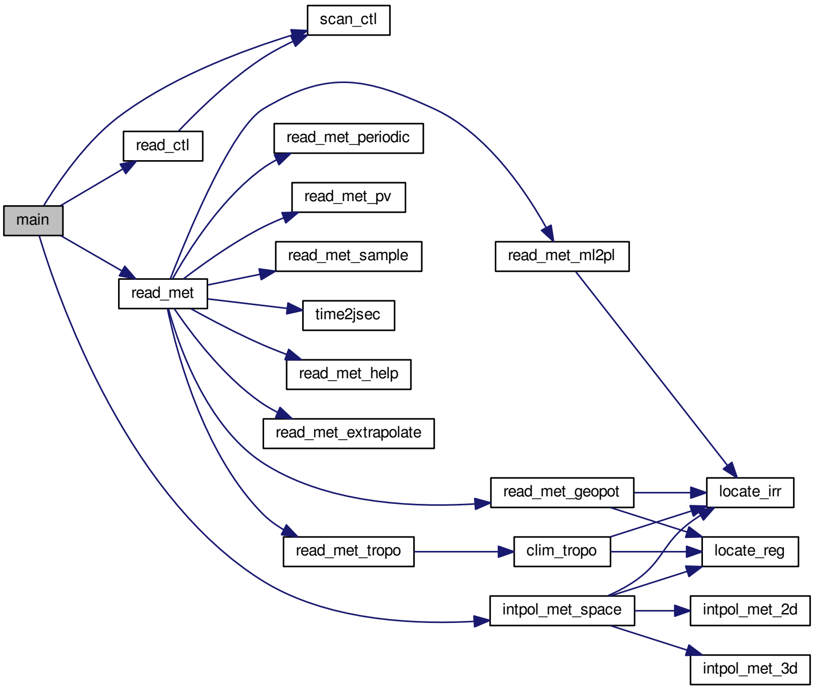
5.23.2 Function Documentation

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

Definition at line 27 of file met_map.c.

```
                {

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[EX][EY], p0, ps, psm[EX][EY], pt, ptm[EX][EY], t,
    tm[EX][EY], u, um[EX][EY], v, vm[EX][EY], w, wm[EX][EY], h2o,
    h2om[EX][EY], o3, o3m[EX][EY], z, zm[EX][EY], pv, pvm[EX][EY], zt,
    ztm[EX][EY], tt, ttm[EX][EY], lon, lon0, lon1, lons[EX], dlon, lat, lat0,
    lat1, lats[EY], dlat;

  static int i, ix, iy, np[EX][EY], nx, ny;

  /* Allocate... */
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "", NULL));
  lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);

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

    /* Read meteorological data... */
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Set horizontal grid... */
    if (dlon <= 0)
      dlon = fabs(met->lon[1] - met->lon[0]);
    if (dlat <= 0)
      dlat = fabs(met->lat[1] - met->lat[0]);
    if (lon0 < -360 && lon1 > 360) {
      lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
      lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
    }
    nx = ny = 0;
    for (lon = lon0; lon <= lon1; lon += dlon) {
      lons[nx] = lon;
      if ((++nx) > EX)
        ERRMSG("Too many longitudes!");
    }
    if (lat0 < -90 && lat1 > 90) {
      lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
      lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
    }
    for (lat = lat0; lat <= lat1; lat += dlat) {
      lats[ny] = lat;
      if ((++ny) > EY)
        ERRMSG("Too many latitudes!");
    }

    /* Average data... */
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {

        /* Interpolate to given log-pressure height... */
        intpol_met_space(met, p0, lons[ix], lats[iy], &ps, &pt,
                         &z, &t, &u, &v, &w, &pv, &h2o, &o3);

        /* Get tropopause data... */
        intpol_met_space(met, pt, lons[ix], lats[iy], NULL, NULL,
                         &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);

        /* Sum up data... */
        timem[ix][iy] += met->time;
        zm[ix][iy] += z;
        tm[ix][iy] += t;
        um[ix][iy] += u;
        vm[ix][iy] += v;
        wm[ix][iy] += w;
        pvm[ix][iy] += pv;
        h2om[ix][iy] += h2o;
        o3m[ix][iy] += o3;
        psm[ix][iy] += ps;
        ptm[ix][iy] += pt;
        ztm[ix][iy] += zt;
        ttm[ix][iy] += tt;
        np[ix][iy]++;
      }
  }

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n"
          "# $10 = H2O volume mixing ratio [1]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n");

  /* Write data... */
  for (iy = 0; iy < ny; iy++) {
    fprintf(out, "\n");
    for (ix = 0; ix < nx; ix++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
              tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
              vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
              h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
              zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
              psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
              ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
  }

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.24 met_map.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[EX][EY], p0, ps, psm[EX][EY], pt, ptm[EX][EY], t,
    tm[EX][EY], u, um[EX][EY], v, vm[EX][EY], w, wm[EX][EY], h2o,
    h2om[EX][EY], o3, o3m[EX][EY], z, zm[EX][EY], pv, pvm[EX][EY], zt,
    ztm[EX][EY], tt, ttm[EX][EY], lon, lon0, lon1, lons[EX], dlon, lat, lat0,
    lat1, lats[EY], dlat;

  static int i, ix, iy, np[EX][EY], nx, ny;

  /* Allocate... */
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "", NULL));
  lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);

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

    /* Read meteorological data... */
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Set horizontal grid... */
    if (dlon <= 0)
      dlon = fabs(met->lon[1] - met->lon[0]);
    if (dlat <= 0)
      dlat = fabs(met->lat[1] - met->lat[0]);
    if (lon0 < -360 && lon1 > 360) {
      lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
      lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
    }
    nx = ny = 0;
    for (lon = lon0; lon <= lon1; lon += dlon) {
      lons[nx] = lon;
      if ((++nx) > EX)
        ERRMSG("Too many longitudes!");
    }
    if (lat0 < -90 && lat1 > 90) {
      lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
      lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
    }
    for (lat = lat0; lat <= lat1; lat += dlat) {
      lats[ny] = lat;
      if ((++ny) > EY)
        ERRMSG("Too many latitudes!");
    }

    /* Average data... */
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {

        /* Interpolate to given log-pressure height... */
        intpol_met_space(met, p0, lons[ix], lats[iy], &ps, &pt,
                         &z, &t, &u, &v, &w, &pv, &h2o, &o3);

        /* Get tropopause data... */
        intpol_met_space(met, pt, lons[ix], lats[iy], NULL, NULL,
                         &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);

        /* Sum up data... */
        timem[ix][iy] += met->time;
        zm[ix][iy] += z;
        tm[ix][iy] += t;
        um[ix][iy] += u;
        vm[ix][iy] += v;
        wm[ix][iy] += w;
        pvm[ix][iy] += pv;
        h2om[ix][iy] += h2o;
        o3m[ix][iy] += o3;
        psm[ix][iy] += ps;
        ptm[ix][iy] += pt;
        ztm[ix][iy] += zt;
        ttm[ix][iy] += tt;
        np[ix][iy]++;
      }
  }

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n"
          "# $10 = H2O volume mixing ratio [1]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n");

  /* Write data... */
  for (iy = 0; iy < ny; iy++) {
    fprintf(out, "\n");
    for (ix = 0; ix < nx; ix++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
              tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
              vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
              h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
              zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
              psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
              ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy]);
  }

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

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

  return EXIT_SUCCESS;
}
```

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.

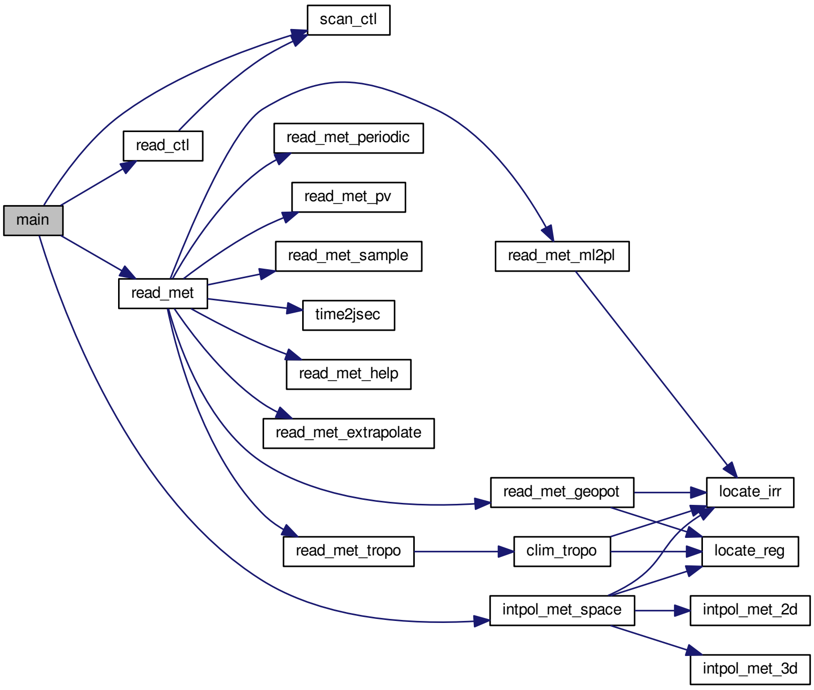
5.25.2 Function Documentation

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

Definition at line 38 of file met_prof.c.

```
                {

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
    lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
    wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
    zg, zgm[NZ], zt, ztm[NZ], pv, pvm[NZ];

  static int i, iz, np[NZ];

  /* Allocate... */
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "60", NULL);
  dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "1", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);

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

    /* Read meteorological data... */
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Set horizontal grid spacing... */
    if (dlon <= 0)
      dlon = fabs(met->lon[1] - met->lon[0]);
    if (dlat <= 0)
      dlat = fabs(met->lat[1] - met->lat[0]);

    /* Average... */
    for (z = z0; z <= z1; z += dz) {
      iz = (int) ((z - z0) / dz);
      if (iz < 0 || iz > NZ)
        ERRMSG("Too many altitudes!");
      for (lon = lon0; lon <= lon1; lon += dlon)
        for (lat = lat0; lat <= lat1; lat += dlat) {
          intpol_met_space(met, P(z), lon, lat, &ps, &pt, &zg,
                           &t, &u, &v, &w, &pv, &h2o, &o3);
          intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
                           &tt, NULL, NULL, NULL, NULL, NULL, NULL);
          if (gsl_finite(t) && gsl_finite(u)
              && gsl_finite(v) && gsl_finite(w)) {
            timem[iz] += met->time;
            lonm[iz] += lon;
            latm[iz] += lat;
            zgm[iz] += zg;
            tm[iz] += t;
            um[iz] += u;
            vm[iz] += v;
            wm[iz] += w;
            pvm[iz] += pv;
            h2om[iz] += h2o;
            o3m[iz] += o3;
            psm[iz] += ps;
            ptm[iz] += pt;
            ztm[iz] += zt;
            ttm[iz] += tt;
            np[iz]++;
          }
        }
    }
  }

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n");
  fprintf(out,
          "# $10 = H2O volume mixing ratio [1]\n"
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n\n");

  /* Write data... */
  for (z = z0; z <= z1; z += dz) {
    iz = (int) ((z - z0) / dz);
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
            timem[iz] / np[iz], z, lonm[iz] / np[iz], latm[iz] / np[iz], P(z),
            tm[iz] / np[iz], um[iz] / np[iz], vm[iz] / np[iz],
            wm[iz] / np[iz], h2om[iz] / np[iz], o3m[iz] / np[iz],
            zgm[iz] / np[iz], pvm[iz] / np[iz], psm[iz] / np[iz],
            ptm[iz] / np[iz], ztm[iz] / np[iz], ttm[iz] / np[iz]);
  }

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.26 met_prof.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum number of altitudes. */
#define NZ 1000

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

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

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
    lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
    wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ], pt, ptm[NZ], tt, ttm[NZ],
    zg, zgm[NZ], zt, ztm[NZ], pv, pvm[NZ];

  static int i, iz, np[NZ];

  /* Allocate... */
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "60", NULL);
  dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "1", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);

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

    /* Read meteorological data... */
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Set horizontal grid spacing... */
    if (dlon <= 0)
      dlon = fabs(met->lon[1] - met->lon[0]);
    if (dlat <= 0)
      dlat = fabs(met->lat[1] - met->lat[0]);

    /* Average... */
    for (z = z0; z <= z1; z += dz) {
      iz = (int) ((z - z0) / dz);
      if (iz < 0 || iz > NZ)
        ERRMSG("Too many altitudes!");
      for (lon = lon0; lon <= lon1; lon += dlon)
        for (lat = lat0; lat <= lat1; lat += dlat) {
          intpol_met_space(met, P(z), lon, lat, &ps, &pt, &zg,
                           &t, &u, &v, &w, &pv, &h2o, &o3);
          intpol_met_space(met, pt, lon, lat, NULL, NULL, &zt,
                           &tt, NULL, NULL, NULL, NULL, NULL, NULL);
          if (gsl_finite(t) && gsl_finite(u)
              && gsl_finite(v) && gsl_finite(w)) {
            timem[iz] += met->time;
            lonm[iz] += lon;
            latm[iz] += lat;
            zgm[iz] += zg;
            tm[iz] += t;
            um[iz] += u;
            vm[iz] += v;
            wm[iz] += w;
            pvm[iz] += pv;
            h2om[iz] += h2o;
            o3m[iz] += o3;
            psm[iz] += ps;
            ptm[iz] += pt;
            ztm[iz] += zt;
            ttm[iz] += tt;
            np[iz]++;
          }
        }
    }
  }

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n");
  fprintf(out,
          "# $10 = H2O volume mixing ratio [1]\n"
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n\n");

  /* Write data... */
  for (z = z0; z <= z1; z += dz) {
    iz = (int) ((z - z0) / dz);
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
            timem[iz] / np[iz], z, lonm[iz] / np[iz], latm[iz] / np[iz], P(z),
            tm[iz] / np[iz], um[iz] / np[iz], vm[iz] / np[iz],
            wm[iz] / np[iz], h2om[iz] / np[iz], o3m[iz] / np[iz],
            zgm[iz] / np[iz], pvm[iz] / np[iz], psm[iz] / np[iz],
            ptm[iz] / np[iz], ztm[iz] / np[iz], ttm[iz] / np[iz]);
  }

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

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

  return EXIT_SUCCESS;
}
```

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.

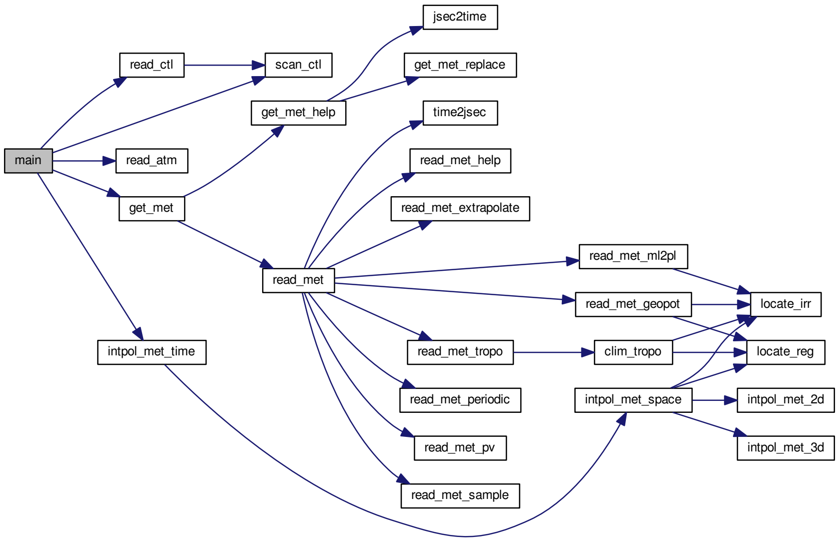
5.27.2 Function Documentation

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

Definition at line 31 of file met_sample.c.

```
                {

  ctl_t ctl;

  atm_t *atm;

  met_t *met0, *met1;

  FILE *out;

  double h2o, o3, p0, p1, pref, ps, pt, pv, t, tt, u, v, w, z, zm, zref, zt;

  int geopot, ip, it;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");

  /* Allocate... */
  ALLOC(atm, atm_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  geopot =
    (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);

  /* Read atmospheric data... */
  if (!read_atm(argv[4], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n");
  fprintf(out,
          "# $10 = H2O volume mixing ratio [1]\n"
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n\n");

  /* Loop over air parcels... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Get meteorological data... */
    get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);

    /* Set reference pressure for interpolation... */
    pref = atm->p[ip];
    if (geopot) {
      zref = Z(pref);
      p0 = met0->p[0];
      p1 = met0->p[met0->np - 1];
      for (it = 0; it < 24; it++) {
        pref = 0.5 * (p0 + p1);
        intpol_met_time(met0, met1, atm->time[ip], pref, atm->
lon[ip],
                        atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
                        NULL, NULL, NULL);
        if (zref > zm || !gsl_finite(zm))
          p0 = pref;
        else
          p1 = pref;
      }
      pref = 0.5 * (p0 + p1);
    }

    /* Interpolate meteorological data... */
    intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
                    atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
                    &o3);
    intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
lat[ip],
                    NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);

    /* Write data... */
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
            atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
  }

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

  /* Free... */
  free(atm);
  free(met0);
  free(met1);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.28 met_sample.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

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

  ctl_t ctl;

  atm_t *atm;

  met_t *met0, *met1;

  FILE *out;

  double h2o, o3, p0, p1, pref, ps, pt, pv, t, tt, u, v, w, z, zm, zref, zt;

  int geopot, ip, it;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");

  /* Allocate... */
  ALLOC(atm, atm_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  geopot =
    (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);

  /* Read atmospheric data... */
  if (!read_atm(argv[4], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n");
  fprintf(out,
          "# $10 = H2O volume mixing ratio [1]\n"
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n\n");

  /* Loop over air parcels... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Get meteorological data... */
    get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);

    /* Set reference pressure for interpolation... */
    pref = atm->p[ip];
    if (geopot) {
      zref = Z(pref);
      p0 = met0->p[0];
      p1 = met0->p[met0->np - 1];
      for (it = 0; it < 24; it++) {
        pref = 0.5 * (p0 + p1);
        intpol_met_time(met0, met1, atm->time[ip], pref, atm->
lon[ip],
                        atm->lat[ip], NULL, NULL, &zm, NULL, NULL, NULL, NULL,
                        NULL, NULL, NULL);
        if (zref > zm || !gsl_finite(zm))
          p0 = pref;
        else
          p1 = pref;
      }
      pref = 0.5 * (p0 + p1);
    }

    /* Interpolate meteorological data... */
    intpol_met_time(met0, met1, atm->time[ip], pref, atm->lon[ip],
                    atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o,
                    &o3);
    intpol_met_time(met0, met1, atm->time[ip], pt, atm->lon[ip], atm->
lat[ip],
                    NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL, NULL);

    /* Write data... */
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
            atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt);
  }

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

  /* Free... */
  free(atm);
  free(met0);
  free(met1);

  return EXIT_SUCCESS;
}
```

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.

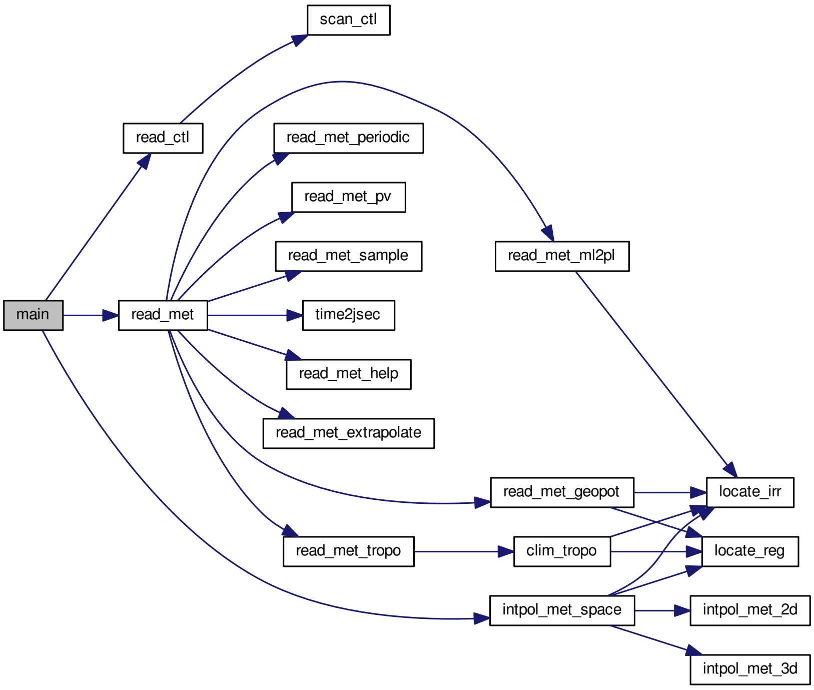
5.29.2 Function Documentation

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

Definition at line 27 of file met_zm.c.

```
                {

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], ttm[EP][EY],
    ztm[EP][EY], tm[EP][EY], um[EP][EY], vm[EP][EY], wm[EP][EY], h2om[EP][EY],
    pvm[EP][EY], o3m[EP][EY], zm[EP][EY], zt, tt;

  static int i, ip, ix, iy, np[EP][EY], npt[EP][EY];

  /* Allocate... */
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);

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

    /* Read meteorological data... */
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Average data... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        for (ip = 0; ip < met->np; ip++) {
          intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->lat[iy],
                           NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
                           NULL);
          timem[ip][iy] += met->time;
          zm[ip][iy] += met->z[ix][iy][ip];
          tm[ip][iy] += met->t[ix][iy][ip];
          um[ip][iy] += met->u[ix][iy][ip];
          vm[ip][iy] += met->v[ix][iy][ip];
          wm[ip][iy] += met->w[ix][iy][ip];
          pvm[ip][iy] += met->pv[ix][iy][ip];
          h2om[ip][iy] += met->h2o[ix][iy][ip];
          o3m[ip][iy] += met->o3[ix][iy][ip];
          psm[ip][iy] += met->ps[ix][iy];
          if (gsl_finite(met->pt[ix][iy])) {
            ptm[ip][iy] += met->pt[ix][iy];
            ztm[ip][iy] += zt;
            ttm[ip][iy] += tt;
            npt[ip][iy]++;
          }
          np[ip][iy]++;
        }
  }

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n"
          "# $10 = H2O volume mixing ratio [1]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n");

  /* Write data... */
  for (ip = 0; ip < met->np; ip++) {
    fprintf(out, "\n");
    for (iy = 0; iy < met->ny; iy++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              timem[ip][iy] / np[ip][iy], Z(met->p[ip]), 0.0, met->lat[iy],
              met->p[ip], tm[ip][iy] / np[ip][iy], um[ip][iy] / np[ip][iy],
              vm[ip][iy] / np[ip][iy], wm[ip][iy] / np[ip][iy],
              h2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
              zm[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
              psm[ip][iy] / np[ip][iy], ptm[ip][iy] / npt[ip][iy],
              ztm[ip][iy] / npt[ip][iy], ttm[ip][iy] / npt[ip][iy]);
  }

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.30 met_zm.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[EP][EY], psm[EP][EY], ptm[EP][EY], ttm[EP][EY],
    ztm[EP][EY], tm[EP][EY], um[EP][EY], vm[EP][EY], wm[EP][EY], h2om[EP][EY],
    pvm[EP][EY], o3m[EP][EY], zm[EP][EY], zt, tt;

  static int i, ip, ix, iy, np[EP][EY], npt[EP][EY];

  /* Allocate... */
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);

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

    /* Read meteorological data... */
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Average data... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        for (ip = 0; ip < met->np; ip++) {
          intpol_met_space(met, met->pt[ix][iy], met->lon[ix], met->
lat[iy],
                           NULL, NULL, &zt, &tt, NULL, NULL, NULL, NULL, NULL,
                           NULL);
          timem[ip][iy] += met->time;
          zm[ip][iy] += met->z[ix][iy][ip];
          tm[ip][iy] += met->t[ix][iy][ip];
          um[ip][iy] += met->u[ix][iy][ip];
          vm[ip][iy] += met->v[ix][iy][ip];
          wm[ip][iy] += met->w[ix][iy][ip];
          pvm[ip][iy] += met->pv[ix][iy][ip];
          h2om[ip][iy] += met->h2o[ix][iy][ip];
          o3m[ip][iy] += met->o3[ix][iy][ip];
          psm[ip][iy] += met->ps[ix][iy];
          if (gsl_finite(met->pt[ix][iy])) {
            ptm[ip][iy] += met->pt[ix][iy];
            ztm[ip][iy] += zt;
            ttm[ip][iy] += tt;
            npt[ip][iy]++;
          }
          np[ip][iy]++;
        }
  }

  /* Create output file... */
  printf("Write meteorological data file: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1   = time [s]\n"
          "# $2   = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = zonal wind [m/s]\n"
          "# $8  = meridional wind [m/s]\n"
          "# $9  = vertical wind [hPa/s]\n"
          "# $10 = H2O volume mixing ratio [1]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [1]\n"
          "# $12 = geopotential height [km]\n"
          "# $13 = potential vorticity [PVU]\n"
          "# $14 = surface pressure [hPa]\n"
          "# $15 = tropopause pressure [hPa]\n"
          "# $16 = tropopause geopotential height [km]\n"
          "# $17 = tropopause temperature [K]\n");

  /* Write data... */
  for (ip = 0; ip < met->np; ip++) {
    fprintf(out, "\n");
    for (iy = 0; iy < met->ny; iy++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              timem[ip][iy] / np[ip][iy], Z(met->p[ip]), 0.0, met->lat[iy],
              met->p[ip], tm[ip][iy] / np[ip][iy], um[ip][iy] / np[ip][iy],
              vm[ip][iy] / np[ip][iy], wm[ip][iy] / np[ip][iy],
              h2om[ip][iy] / np[ip][iy], o3m[ip][iy] / np[ip][iy],
              zm[ip][iy] / np[ip][iy], pvm[ip][iy] / np[ip][iy],
              psm[ip][iy] / np[ip][iy], ptm[ip][iy] / npt[ip][iy],
              ztm[ip][iy] / npt[ip][iy], ttm[ip][iy] / npt[ip][iy]);
  }

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

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

  return EXIT_SUCCESS;
}
```

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.

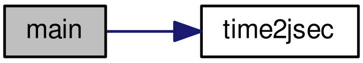
5.31.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

```
                {

  double jsec, remain;

  int day, hour, min, mon, sec, year;

  /* Check arguments... */
  if (argc < 8)
    ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");

  /* Read arguments... */
  year = atoi(argv[1]);
  mon = atoi(argv[2]);
  day = atoi(argv[3]);
  hour = atoi(argv[4]);
  min = atoi(argv[5]);
  sec = atoi(argv[6]);
  remain = atof(argv[7]);

  /* Convert... */
  time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
  printf("%.2f\n", jsec);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.32 time2jsec.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  double jsec, remain;

  int day, hour, min, mon, sec, year;

  /* Check arguments... */
  if (argc < 8)
    ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");

  /* Read arguments... */
  year = atoi(argv[1]);
  mon = atoi(argv[2]);
  day = atoi(argv[3]);
  hour = atoi(argv[4]);
  min = atoi(argv[5]);
  sec = atoi(argv[6]);
  remain = atof(argv[7]);

  /* Convert... */
  time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
  printf("%.2f\n", jsec);

  return EXIT_SUCCESS;
}
```

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

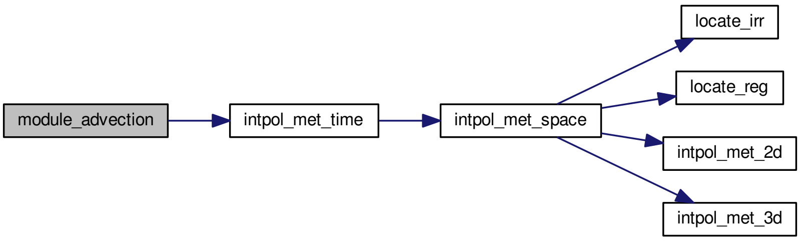
5.33.2.1 void module_advection (met_t ∗ *met0*, met_t ∗ *met1*, atm_t ∗ *atm*, int *ip*, double *dt*)

Calculate advection of air parcels.

Definition at line 387 of file trac.c.

```
                  {

  double v[3], xm[3];

  /* Interpolate meteorological data... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                  atm->lon[ip], atm->lat[ip], NULL, NULL, NULL,
                  &v[0], &v[1], &v[2], NULL, NULL, NULL);

  /* Get position of the mid point... */
  xm[0] = atm->lon[ip] + DX2DEG(0.5 * dt * v[0] / 1000., atm->lat[ip]);
  xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt * v[1] / 1000.);
  xm[2] = atm->p[ip] + 0.5 * dt * v[2];

  /* Interpolate meteorological data for mid point... */
  intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
                  xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
                  &v[0], &v[1], &v[2], NULL, NULL, NULL);

  /* Save new position... */
  atm->time[ip] += dt;
  atm->lon[ip] += DX2DEG(dt * v[0] / 1000., xm[1]);
  atm->lat[ip] += DY2DEG(dt * v[1] / 1000.);
  atm->p[ip] += dt * v[2];
}
```

Here is the call graph for this function:



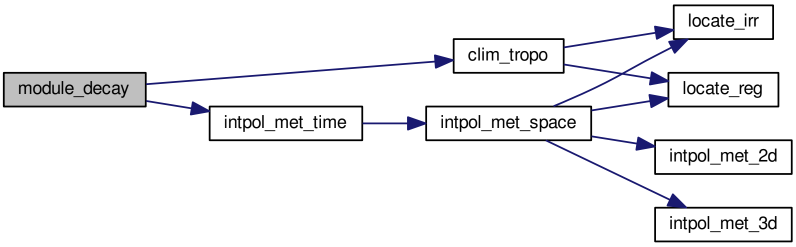
5.33.2.2 void module_decay (ctl_t ∗ *ctl*, met_t ∗ *met0*, met_t ∗ *met1*, atm_t ∗ *atm*, int *ip*, double *dt*)

Calculate exponential decay of particle mass.

Definition at line 420 of file trac.c.

```
                  {

  double ps, pt, tdec;

  /* Set constant lifetime... */
  if (ctl->tdec_trop == ctl->tdec_strat)
    tdec = ctl->tdec_trop;

  /* Set altitude-dependent lifetime... */
  else {

    /* Get surface pressure... */
    intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                    atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
                    NULL, NULL, NULL, NULL, NULL, NULL);

    /* Get tropopause pressure... */
    pt = clim_tropo(atm->time[ip], atm->lat[ip]);

    /* Set lifetime... */
    if (atm->p[ip] <= pt)
      tdec = ctl->tdec_strat;
    else
      tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->p[ip]);
  }

  /* Calculate exponential decay... */
  atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
}
```

Here is the call graph for this function:



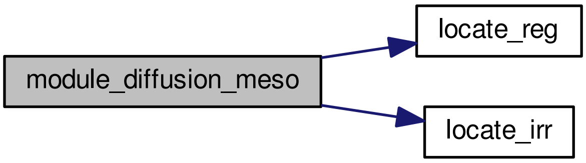
5.33.2.3 void module_diffusion_meso (ctl_t ∗ *ctl*, met_t ∗ *met0*, met_t ∗ *met1*, atm_t ∗ *atm*, int *ip*, double *dt*, gsl_rng ∗ *rng*)

Calculate mesoscale diffusion.

Definition at line 458 of file trac.c.

```
                 {

  double r, rs, u[16], v[16], w[16];

  int ix, iy, iz;

  /* Get indices... */
  ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
  iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
  iz = locate_irr(met0->p, met0->np, atm->p[ip]);

  /* Caching of wind standard deviations... */
  if (atm->cache_time[ix][iy][iz] != met0->time) {

    /* Collect local wind data... */
    u[0] = met0->u[ix][iy][iz];
    u[1] = met0->u[ix + 1][iy][iz];
    u[2] = met0->u[ix][iy + 1][iz];
    u[3] = met0->u[ix + 1][iy + 1][iz];
    u[4] = met0->u[ix][iy][iz + 1];
    u[5] = met0->u[ix + 1][iy][iz + 1];
    u[6] = met0->u[ix][iy + 1][iz + 1];
    u[7] = met0->u[ix + 1][iy + 1][iz + 1];

    v[0] = met0->v[ix][iy][iz];
    v[1] = met0->v[ix + 1][iy][iz];
    v[2] = met0->v[ix][iy + 1][iz];
    v[3] = met0->v[ix + 1][iy + 1][iz];
    v[4] = met0->v[ix][iy][iz + 1];
    v[5] = met0->v[ix + 1][iy][iz + 1];
    v[6] = met0->v[ix][iy + 1][iz + 1];
    v[7] = met0->v[ix + 1][iy + 1][iz + 1];

    w[0] = met0->w[ix][iy][iz];
    w[1] = met0->w[ix + 1][iy][iz];
    w[2] = met0->w[ix][iy + 1][iz];
    w[3] = met0->w[ix + 1][iy + 1][iz];
    w[4] = met0->w[ix][iy][iz + 1];
    w[5] = met0->w[ix + 1][iy][iz + 1];
    w[6] = met0->w[ix][iy + 1][iz + 1];
    w[7] = met0->w[ix + 1][iy + 1][iz + 1];

    /* Collect local wind data... */
    u[8] = met1->u[ix][iy][iz];
    u[9] = met1->u[ix + 1][iy][iz];
    u[10] = met1->u[ix][iy + 1][iz];
    u[11] = met1->u[ix + 1][iy + 1][iz];
    u[12] = met1->u[ix][iy][iz + 1];
    u[13] = met1->u[ix + 1][iy][iz + 1];
    u[14] = met1->u[ix][iy + 1][iz + 1];
    u[15] = met1->u[ix + 1][iy + 1][iz + 1];

    v[8] = met1->v[ix][iy][iz];
    v[9] = met1->v[ix + 1][iy][iz];
    v[10] = met1->v[ix][iy + 1][iz];
    v[11] = met1->v[ix + 1][iy + 1][iz];
    v[12] = met1->v[ix][iy][iz + 1];
    v[13] = met1->v[ix + 1][iy][iz + 1];
    v[14] = met1->v[ix][iy + 1][iz + 1];
    v[15] = met1->v[ix + 1][iy + 1][iz + 1];

    w[8] = met1->w[ix][iy][iz];
    w[9] = met1->w[ix + 1][iy][iz];
    w[10] = met1->w[ix][iy + 1][iz];
    w[11] = met1->w[ix + 1][iy + 1][iz];
    w[12] = met1->w[ix][iy][iz + 1];
    w[13] = met1->w[ix + 1][iy][iz + 1];
    w[14] = met1->w[ix][iy + 1][iz + 1];
    w[15] = met1->w[ix + 1][iy + 1][iz + 1];

    /* Get standard deviations of local wind data... */
    atm->cache_usig[ix][iy][iz] = (float) gsl_stats_sd(u, 1, 16);
    atm->cache_vsig[ix][iy][iz] = (float) gsl_stats_sd(v, 1, 16);
    atm->cache_wsig[ix][iy][iz] = (float) gsl_stats_sd(w, 1, 16);
    atm->cache_time[ix][iy][iz] = met0->time;
  }

  /* Set temporal correlations for mesoscale fluctuations... */
  r = 1 - 2 * fabs(dt) / ctl->dt_met;
  rs = sqrt(1 - r * r);

  /* Calculate horizontal mesoscale wind fluctuations... */
  if (ctl->turb_mesox > 0) {
    atm->up[ip] = (float)
      (r * atm->up[ip]
       + rs * gsl_ran_gaussian_ziggurat(rng,
                                        ctl->turb_mesox *
                                        atm->cache_usig[ix][iy][iz]));
    atm->lon[ip] += DX2DEG(atm->up[ip] * dt / 1000., atm->lat[ip]);

    atm->vp[ip] = (float)
      (r * atm->vp[ip]
       + rs * gsl_ran_gaussian_ziggurat(rng,
                                        ctl->turb_mesox *
                                        atm->cache_vsig[ix][iy][iz]));
    atm->lat[ip] += DY2DEG(atm->vp[ip] * dt / 1000.);
  }

  /* Calculate vertical mesoscale wind fluctuations... */
  if (ctl->turb_mesoz > 0) {
    atm->wp[ip] = (float)
      (r * atm->wp[ip]
       + rs * gsl_ran_gaussian_ziggurat(rng,
                                        ctl->turb_mesoz *
                                        atm->cache_wsig[ix][iy][iz]));
    atm->p[ip] += atm->wp[ip] * dt;
  }
}
```

Here is the call graph for this function:



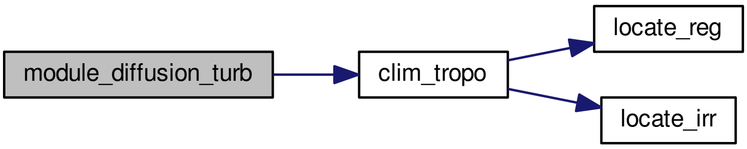
5.33.2.4 void module_diffusion_turb (ctl_t ∗ *ctl,* atm_t ∗ *atm,* int *ip,* double *dt,* gsl_rng ∗ *rng*)

Calculate turbulent diffusion.

Definition at line 576 of file trac.c.

```
                  {

  double dx, dz, pt, p0, p1, w;

  /* Get tropopause pressure... */
  pt = clim_tropo(atm->time[ip], atm->lat[ip]);

  /* Get weighting factor... */
  p1 = pt * 0.866877899;
  p0 = pt / 0.866877899;
  if (atm->p[ip] > p0)
    w = 1;
  else if (atm->p[ip] < p1)
    w = 0;
  else
    w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);

  /* Set diffusivity... */
  dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
  dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;

  /* Horizontal turbulent diffusion... */
  if (dx > 0) {
    atm->lon[ip]
      += DX2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
                / 1000., atm->lat[ip]);
    atm->lat[ip]
      += DY2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
                / 1000.);
  }

  /* Vertical turbulent diffusion... */
  if (dz > 0)
    atm->p[ip]
      += DZ2DP(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
               / 1000., atm->p[ip]);
}
```

Here is the call graph for this function:



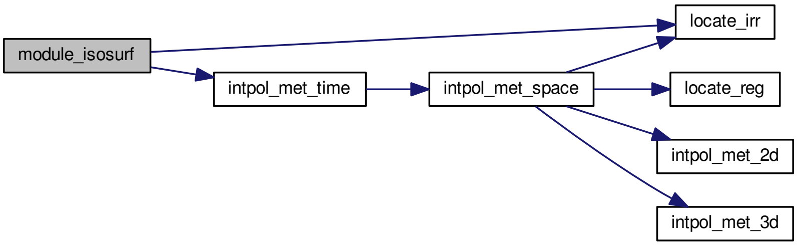
5.33.2.5 void module_isosurf (ctl_t ∗ *ctl*, met_t ∗ *met0*, met_t ∗ *met1*, atm_t ∗ *atm*, int *ip*)

Force air parcels to stay on isosurface.

Definition at line 621 of file trac.c.

```
        {

static double *iso, *ps, t, *ts;

static int idx, ip2, n;

FILE *in;

char line[LEN];

/* Initialize... */
if (ip < 0) {

  /* Allocate... */
  ALLOC(iso, double,
        NP);
  ALLOC(ps, double,
        NP);
  ALLOC(ts, double,
        NP);

  /* Save pressure... */
  if (ctl->isosurf == 1)
    for (ip2 = 0; ip2 < atm->np; ip2++)
      iso[ip2] = atm->p[ip2];

  /* Save density... */
  else if (ctl->isosurf == 2)
    for (ip2 = 0; ip2 < atm->np; ip2++) {
      intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                      atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
                      &t, NULL, NULL, NULL, NULL, NULL, NULL);
      iso[ip2] = atm->p[ip2] / t;
    }

  /* Save potential temperature... */
  else if (ctl->isosurf == 3)
    for (ip2 = 0; ip2 < atm->np; ip2++) {
      intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                      atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
                      &t, NULL, NULL, NULL, NULL, NULL, NULL);
      iso[ip2] = THETA(atm->p[ip2], t);
    }

  /* Read balloon pressure data... */
  else if (ctl->isosurf == 4) {

    /* Write info... */
    printf("Read balloon pressure data: %s\n", ctl->balloon);

    /* Open file... */
    if (!(in = fopen(ctl->balloon, "r")))
      ERRMSG("Cannot open file!");

      /* Read pressure time series... */
      while (fgets(line, LEN, in))
        if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
          if ((++n) > NP)
            ERRMSG("Too many data points!");

      /* Check number of points... */
      if (n < 1)
        ERRMSG("Could not read any data!");

      /* Close file... */
      fclose(in);
    }

    /* Leave initialization... */
    return;
  }

  /* Restore pressure... */
  if (ctl->isosurf == 1)
    atm->p[ip] = iso[ip];

  /* Restore density... */
  else if (ctl->isosurf == 2) {
    intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->lon[ip],
                    atm->lat[ip], NULL, NULL, NULL, &t,
                    NULL, NULL, NULL, NULL, NULL, NULL);
    atm->p[ip] = iso[ip] * t;
  }

  /* Restore potential temperature... */
  else if (ctl->isosurf == 3) {
    intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->lon[ip],
                    atm->lat[ip], NULL, NULL, NULL, &t,
                    NULL, NULL, NULL, NULL, NULL, NULL);
    atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
  }

  /* Interpolate pressure... */
  else if (ctl->isosurf == 4) {
    if (atm->time[ip] <= ts[0])
      atm->p[ip] = ps[0];
    else if (atm->time[ip] >= ts[n - 1])
      atm->p[ip] = ps[n - 1];
    else {
      idx = locate_irr(ts, n, atm->time[ip]);
      atm->p[ip] = LIN(ts[idx], ps[idx],
                       ts[idx + 1], ps[idx + 1], atm->time[ip]);
    }
  }
}
```

Here is the call graph for this function:



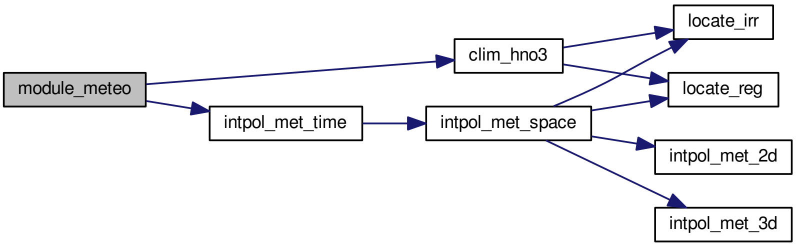
5.33.2.6 void module_meteo (ctl_t ∗ *ctl*, met_t ∗ *met0*, met_t ∗ *met1*, atm_t ∗ *atm*, int *ip*)

Interpolate meteorological data for air parcel positions.

Definition at line 734 of file trac.c.

```
          {

  double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;

  /* Interpolate meteorological data... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->lon[ip],
                  atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);

  /* Set surface pressure... */
  if (ctl->qnt_ps >= 0)
    atm->q[ctl->qnt_ps][ip] = ps;

  /* Set tropopause pressure... */
  if (ctl->qnt_pt >= 0)
    atm->q[ctl->qnt_pt][ip] = pt;

  /* Set pressure... */
  if (ctl->qnt_p >= 0)
    atm->q[ctl->qnt_p][ip] = atm->p[ip];

  /* Set geopotential height... */
  if (ctl->qnt_z >= 0)
    atm->q[ctl->qnt_z][ip] = z;

  /* Set temperature... */
  if (ctl->qnt_t >= 0)
    atm->q[ctl->qnt_t][ip] = t;

  /* Set zonal wind... */
  if (ctl->qnt_u >= 0)
    atm->q[ctl->qnt_u][ip] = u;

  /* Set meridional wind... */
  if (ctl->qnt_v >= 0)
    atm->q[ctl->qnt_v][ip] = v;

  /* Set vertical velocity... */
  if (ctl->qnt_w >= 0)
    atm->q[ctl->qnt_w][ip] = w;

  /* Set water vapor vmr... */
  if (ctl->qnt_h2o >= 0)
    atm->q[ctl->qnt_h2o][ip] = h2o;

  /* Set ozone vmr... */
  if (ctl->qnt_o3 >= 0)
    atm->q[ctl->qnt_o3][ip] = o3;

  /* Calculate horizontal wind... */
  if (ctl->qnt_vh >= 0)
    atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);

  /* Calculate vertical velocity... */
  if (ctl->qnt_vz >= 0)
    atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;

  /* Calculate potential temperature... */
  if (ctl->qnt_theta >= 0)
    atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);

  /* Set potential vorticity... */
  if (ctl->qnt_pv >= 0)
    atm->q[ctl->qnt_pv][ip] = pv;

  /* Calculate T_ice (Marti and Mauersberger, 1993)... */
  if (ctl->qnt_tice >= 0)
    atm->q[ctl->qnt_tice][ip] =
      -2663.5 /
      (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
       12.537);

  /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
  if (ctl->qnt_tnat >= 0) {
    if (ctl->psc_hno3 > 0)
      p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
    else
      p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
        * 1e-9 * atm->p[ip] / 1.333224;
    p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
    a = 0.009179 - 0.00088 * log10(p_h2o);
    b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
    c = -11397.0 / a;
    x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
    x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
    if (x1 > 0)
      atm->q[ctl->qnt_tnat][ip] = x1;
    if (x2 > 0)
      atm->q[ctl->qnt_tnat][ip] = x2;
  }

  /* Calculate T_STS (mean of T_ice and T_NAT)... */
  if (ctl->qnt_tsts >= 0) {
    if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
      ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
    atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
                                       + atm->q[ctl->qnt_tnat][ip]);
  }
}
```

Here is the call graph for this function:



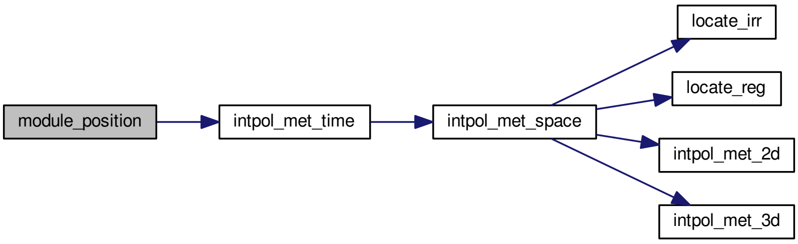
5.33.2.7 void module_position (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

Check position of air parcels.

Definition at line 840 of file trac.c.

```
          {

  double ps;

  /* Calculate modulo... */
  atm->lon[ip] = fmod(atm->lon[ip], 360);
  atm->lat[ip] = fmod(atm->lat[ip], 360);

  /* Check latitude... */
  while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
    if (atm->lat[ip] > 90) {
      atm->lat[ip] = 180 - atm->lat[ip];
      atm->lon[ip] += 180;
    }
    if (atm->lat[ip] < -90) {
      atm->lat[ip] = -180 - atm->lat[ip];
      atm->lon[ip] += 180;
    }
  }

  /* Check longitude... */
  while (atm->lon[ip] < -180)
    atm->lon[ip] += 360;
  while (atm->lon[ip] >= 180)
    atm->lon[ip] -= 360;

  /* Get surface pressure... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                  atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
                  NULL, NULL, NULL, NULL, NULL, NULL);

  /* Check pressure... */
  if (atm->p[ip] > ps)
    atm->p[ip] = ps;
  else if (atm->p[ip] < met0->p[met0->np - 1])
    atm->p[ip] = met0->p[met0->np - 1];
}
```

Here is the call graph for this function:



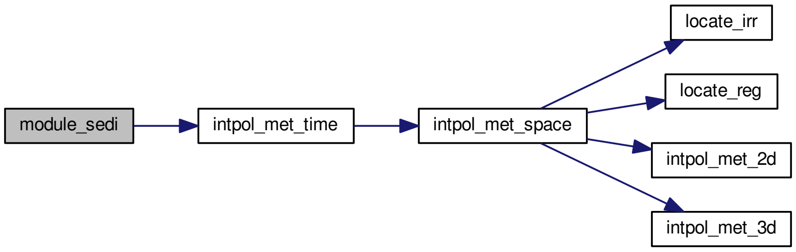
5.33.2.8 void module_sedi (ctl_t ∗ *ctl,* met_t ∗ *met0,* met_t ∗ *met1,* atm_t ∗ *atm,* int *ip,* double *dt*)

Calculate sedimentation of air parcels.

Definition at line 884 of file trac.c.

```
                 {

  /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
  const double A = 1.249, B = 0.42, C = 0.87;

  /* Average mass of an air molecule [kg/molec]: */
  const double m = 4.8096e-26;

  double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;

  /* Convert units... */
  p = 100 * atm->p[ip];
  r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
  rho_p = atm->q[ctl->qnt_rho][ip];

  /* Get temperature... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->lon[ip],
                  atm->lat[ip], NULL, NULL, NULL, &T,
                  NULL, NULL, NULL, NULL, NULL, NULL);

  /* Density of dry air... */
  rho = p / (RA * T);

  /* Dynamic viscosity of air... */
  eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Thermal velocity of an air molecule... */
  v = sqrt(8 * KB * T / (M_PI * m));

  /* Mean free path of an air molecule... */
  lambda = 2 * eta / (rho * v);

  /* Knudsen number for air... */
  K = lambda / r_p;

  /* Cunningham slip-flow correction... */
  G = 1 + K * (A + B * exp(-C / K));

  /* Sedimentation (fall) velocity... */
  v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;

  /* Calculate pressure change... */
  atm->p[ip] += DZ2DP(v_p * dt / 1000., atm->p[ip]);
}
```

Here is the call graph for this function:



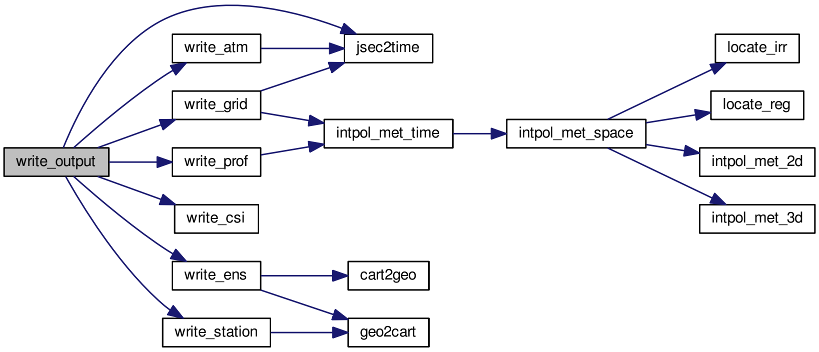
5.33.2.9 void write_output (const char ∗ *dirname*, ctl_t ∗ *ctl*, met_t ∗ *met0*, met_t ∗ *met1*, atm_t ∗ *atm*, double *t*)

Write simulation output.

Definition at line 937 of file trac.c.

```
            {

  char filename[2 * LEN];

  double r;

  int year, mon, day, hour, min, sec;

  /* Get time... */
  jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Write atmospheric data... */
  if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
            dirname, ctl->atm_basename, year, mon, day, hour, min);
    write_atm(filename, ctl, atm, t);
  }

  /* Write CSI data... */
  if (ctl->csi_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
    write_csi(filename, ctl, atm, t);
  }

  /* Write ensemble data... */
  if (ctl->ens_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
    write_ens(filename, ctl, atm, t);
  }

  /* Write gridded data... */
  if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
            dirname, ctl->grid_basename, year, mon, day, hour, min);
    write_grid(filename, ctl, met0, met1, atm, t);
  }

  /* Write profile data... */
  if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write_prof(filename, ctl, met0, met1, atm, t);
  }

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

Here is the call graph for this function:



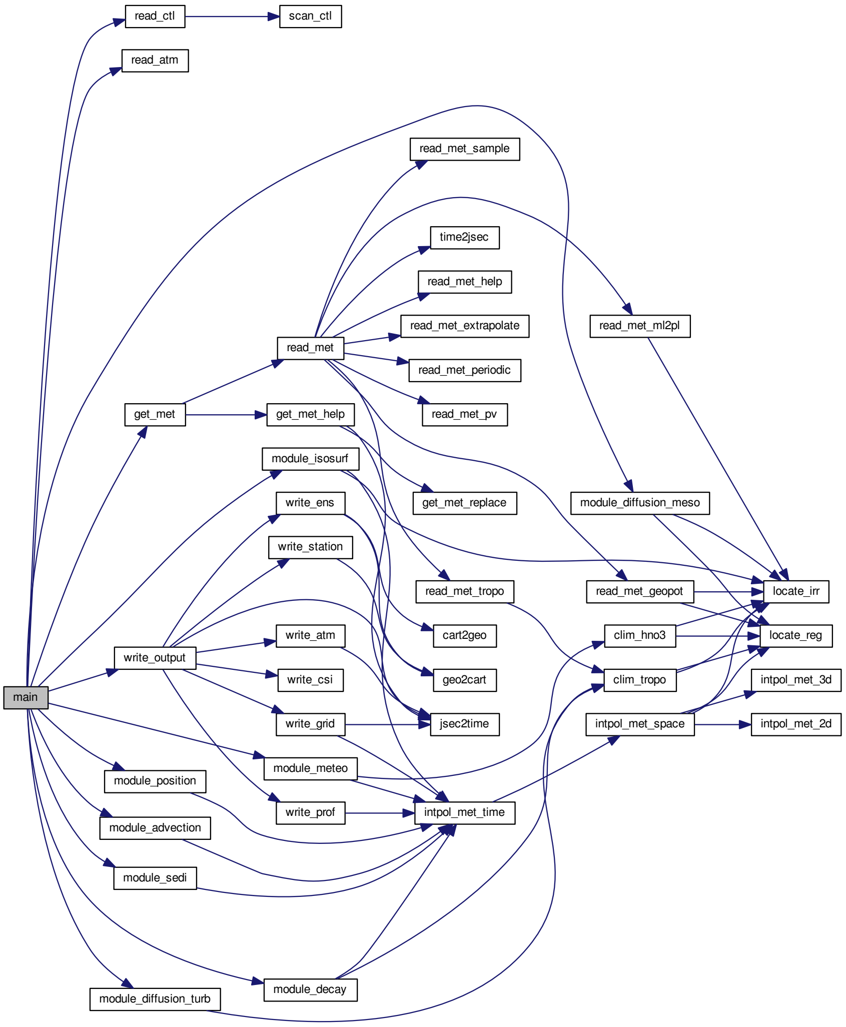
5.33.2.10 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 115 of file trac.c.

```
                {

  ctl_t ctl;

  atm_t *atm;

  met_t *met0, *met1;

  gsl_rng *rng[NTHREADS];

  FILE *dirlist;

  char dirname[LEN], filename[2 * LEN];

  double *dt, t;

  int i, ip, ntask = -1, rank = 0, size = 1;

#ifdef MPI
  /* Initialize MPI... */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
#endif

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");

  /* Open directory list... */
  if (!(dirlist = fopen(argv[1], "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%s", dirname) != EOF) {

    /* MPI parallelization... */
    if ((++ntask) % size != rank)
      continue;

    /* ------------------------------------------------------------
       Initialize model run...
       ------------------------------------------------------------ */

    /* Set timers... */
    START_TIMER(TIMER_TOTAL);
    START_TIMER(TIMER_INIT);

    /* Allocate... */
    ALLOC(atm, atm_t, 1);
    ALLOC(met0, met_t, 1);
    ALLOC(met1, met_t, 1);
    ALLOC(dt, double,
          NP);

    /* Initialize random number generators... */
    gsl_rng_env_setup();
    if (omp_get_max_threads() > NTHREADS)
      ERRMSG("Too many threads!");
    for (i = 0; i < NTHREADS; i++) {
      rng[i] = gsl_rng_alloc(gsl_rng_default);
      gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
    }

    /* Read control parameters... */
    sprintf(filename, "%s/%s", dirname, argv[2]);
    read_ctl(filename, argc, argv, &ctl);

    /* Read atmospheric data... */
    sprintf(filename, "%s/%s", dirname, argv[3]);
    if (!read_atm(filename, &ctl, atm))
      ERRMSG("Cannot open file!");

    /* Set start time... */
    if (ctl.direction == 1) {
      ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
      if (ctl.t_stop > 1e99)
        ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
    } else {
      ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
      if (ctl.t_stop > 1e99)
        ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
    }

    /* Check time interval... */
    if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
      ERRMSG("Nothing to do!");

    /* Round start time... */
    if (ctl.direction == 1)
      ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
    else
      ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;

    /* Set timers... */
    STOP_TIMER(TIMER_INIT);

    /* ------------------------------------------------------------
       Loop over timesteps...
       ------------------------------------------------------------ */

    /* Loop over timesteps... */
    for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
         t += ctl.direction * ctl.dt_mod) {

      /* Adjust length of final time step... */
      if (ctl.direction * (t - ctl.t_stop) > 0)
        t = ctl.t_stop;

      /* Set time steps for air parcels... */
      for (ip = 0; ip < atm->np; ip++)
        if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
             && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
             && ctl.direction * (atm->time[ip] - t) < 0))
          dt[ip] = t - atm->time[ip];
        else
          dt[ip] = GSL_NAN;

      /* Get meteorological data... */
      START_TIMER(TIMER_INPUT);
      get_met(&ctl, argv[4], t, &met0, &met1);
      if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
        printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
               fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
      STOP_TIMER(TIMER_INPUT);

      /* Initialize isosurface... */
      START_TIMER(TIMER_ISOSURF);
      if (ctl.isosurf >= 1 && ctl.isosurf <= 4 && t == ctl.t_start)
        module_isosurf(&ctl, met0, met1, atm, -1);
      STOP_TIMER(TIMER_ISOSURF);

      /* Advection... */
      START_TIMER(TIMER_ADVECT);
```

```
#pragma omp parallel for default(shared) private(ip)
      for (ip = 0; ip < atm->np; ip++)
        if (gsl_finite(dt[ip]))
          module_advection(met0, met1, atm, ip, dt[ip]);
      STOP_TIMER(TIMER_ADVECT);

      /* Turbulent diffusion... */
      START_TIMER(TIMER_DIFFTURB);
      if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
          || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_diffusion_turb(&ctl, atm, ip, dt[ip],
                                  rng[omp_get_thread_num()]);
      }
      STOP_TIMER(TIMER_DIFFTURB);

      /* Mesoscale diffusion... */
      START_TIMER(TIMER_DIFFMESO);
      if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
                                  rng[omp_get_thread_num()]);
      }
      STOP_TIMER(TIMER_DIFFMESO);

      /* Sedimentation... */
      START_TIMER(TIMER_SEDI);
      if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
      }
      STOP_TIMER(TIMER_SEDI);

      /* Isosurface... */
      START_TIMER(TIMER_ISOSURF);
      if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          module_isosurf(&ctl, met0, met1, atm, ip);
      }
      STOP_TIMER(TIMER_ISOSURF);

      /* Position... */
      START_TIMER(TIMER_POSITION);
#pragma omp parallel for default(shared) private(ip)
      for (ip = 0; ip < atm->np; ip++)
        module_position(met0, met1, atm, ip);
      STOP_TIMER(TIMER_POSITION);

      /* Meteorological data... */
      START_TIMER(TIMER_METEO);
      if (ctl.met_dt_out > 0
          && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0)) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          module_meteo(&ctl, met0, met1, atm, ip);
      }
      STOP_TIMER(TIMER_METEO);

      /* Decay... */
      START_TIMER(TIMER_DECAY);
      if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.qnt_m >= 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
      }
      STOP_TIMER(TIMER_DECAY);

      /* Write output... */
      START_TIMER(TIMER_OUTPUT);
      write_output(dirname, &ctl, met0, met1, atm, t);
      STOP_TIMER(TIMER_OUTPUT);
    }

    /* ------------------------------------------------------------
       Finalize model run...
       ------------------------------------------------------------ */

    /* Report memory usage... */
    printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
    printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
    printf("MEMORY_DYNAMIC = %g MByte\n",
           4 * NP * sizeof(double) / 1024. / 1024.);
    printf("MEMORY_STATIC = %g MByte\n",
           ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
            + (EX * EY + EX * EY * EP) * sizeof(float)
            + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);

    /* Report problem size... */
    printf("SIZE_NP = %d\n", atm->np);
    printf("SIZE_TASKS = %d\n", size);
    printf("SIZE_THREADS = %d\n", omp_get_max_threads());

    /* Report timers... */
    STOP_TIMER(TIMER_TOTAL);
    PRINT_TIMER(TIMER_TOTAL);
    PRINT_TIMER(TIMER_INIT);
    PRINT_TIMER(TIMER_INPUT);
    PRINT_TIMER(TIMER_OUTPUT);
    PRINT_TIMER(TIMER_ADVECT);
    PRINT_TIMER(TIMER_DECAY);
    PRINT_TIMER(TIMER_DIFFMESO);
    PRINT_TIMER(TIMER_DIFFTURB);
    PRINT_TIMER(TIMER_ISOSURF);
    PRINT_TIMER(TIMER_METEO);
    PRINT_TIMER(TIMER_POSITION);
    PRINT_TIMER(TIMER_SEDI);

    /* Free random number generators... */
    for (i = 0; i < NTHREADS; i++)
      gsl_rng_free(rng[i]);

    /* Free... */
    free(atm);
    free(met0);
    free(met1);
    free(dt);
  }

#ifdef MPI
  /* Finalize MPI... */
  MPI_Finalize();
#endif

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.34 trac.c

```
/*
  This file is part of MPTRAC.

  MPTRAC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  MPTRAC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

#ifdef MPI
#include "mpi.h"
```

```c
#endif

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

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

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

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

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

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

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

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

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

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

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

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

  ctl_t ctl;

  atm_t *atm;

  met_t *met0, *met1;

  gsl_rng *rng[NTHREADS];

  FILE *dirlist;

  char dirname[LEN], filename[2 * LEN];

  double *dt, t;

  int i, ip, ntask = -1, rank = 0, size = 1;

#ifdef MPI
  /* Initialize MPI... */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
#endif

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");

  /* Open directory list... */
  if (!(dirlist = fopen(argv[1], "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%s", dirname) != EOF) {

    /* MPI parallelization... */
    if ((++ntask) % size != rank)
      continue;

    /* ------------------------------------------------------------
       Initialize model run...
       ------------------------------------------------------------ */

    /* Set timers... */
    START_TIMER(TIMER_TOTAL);
    START_TIMER(TIMER_INIT);

    /* Allocate... */
    ALLOC(atm, atm_t, 1);
    ALLOC(met0, met_t, 1);
    ALLOC(met1, met_t, 1);
    ALLOC(dt, double,
          NP);

    /* Initialize random number generators... */
    gsl_rng_env_setup();
    if (omp_get_max_threads() > NTHREADS)
      ERRMSG("Too many threads!");
    for (i = 0; i < NTHREADS; i++) {
      rng[i] = gsl_rng_alloc(gsl_rng_default);
      gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
    }

    /* Read control parameters... */
    sprintf(filename, "%s/%s", dirname, argv[2]);
    read_ctl(filename, argc, argv, &ctl);

    /* Read atmospheric data... */
    sprintf(filename, "%s/%s", dirname, argv[3]);
    if (!read_atm(filename, &ctl, atm))
      ERRMSG("Cannot open file!");

    /* Set start time... */
    if (ctl.direction == 1) {
      ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
      if (ctl.t_stop > 1e99)
        ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
    } else {
      ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
      if (ctl.t_stop > 1e99)
        ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
    }

    /* Check time interval... */
    if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
      ERRMSG("Nothing to do!");

    /* Round start time... */
    if (ctl.direction == 1)
      ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
    else
      ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;

    /* Set timers... */
    STOP_TIMER(TIMER_INIT);

    /* ------------------------------------------------------------
       Loop over timesteps...
       ------------------------------------------------------------ */

    /* Loop over timesteps... */
    for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
         t += ctl.direction * ctl.dt_mod) {

      /* Adjust length of final time step... */
      if (ctl.direction * (t - ctl.t_stop) > 0)
        t = ctl.t_stop;

      /* Set time steps for air parcels... */
      for (ip = 0; ip < atm->np; ip++)
        if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
             && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
             && ctl.direction * (atm->time[ip] - t) < 0))
          dt[ip] = t - atm->time[ip];
        else
          dt[ip] = GSL_NAN;

      /* Get meteorological data... */
      START_TIMER(TIMER_INPUT);
      get_met(&ctl, argv[4], t, &met0, &met1);
      if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
        printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
               fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
      STOP_TIMER(TIMER_INPUT);

      /* Initialize isosurface... */
      START_TIMER(TIMER_ISOSURF);
      if (ctl.isosurf >= 1 && ctl.isosurf <= 4 && t == ctl.t_start)
        module_isosurf(&ctl, met0, met1, atm, -1);
      STOP_TIMER(TIMER_ISOSURF);

      /* Advection... */
      START_TIMER(TIMER_ADVECT);
#pragma omp parallel for default(shared) private(ip)
      for (ip = 0; ip < atm->np; ip++)
        if (gsl_finite(dt[ip]))
          module_advection(met0, met1, atm, ip, dt[ip]);
      STOP_TIMER(TIMER_ADVECT);

      /* Turbulent diffusion... */
      START_TIMER(TIMER_DIFFTURB);
      if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
          || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_diffusion_turb(&ctl, atm, ip, dt[ip],
                                  rng[omp_get_thread_num()]);
      }
      STOP_TIMER(TIMER_DIFFTURB);

      /* Mesoscale diffusion... */
      START_TIMER(TIMER_DIFFMESO);
      if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
                                  rng[omp_get_thread_num()]);
      }
      STOP_TIMER(TIMER_DIFFMESO);

      /* Sedimentation... */
      START_TIMER(TIMER_SEDI);
      if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
      }
      STOP_TIMER(TIMER_SEDI);

      /* Isosurface... */
      START_TIMER(TIMER_ISOSURF);
      if (ctl.isosurf >= 1 && ctl.isosurf <= 4) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          module_isosurf(&ctl, met0, met1, atm, ip);
      }
      STOP_TIMER(TIMER_ISOSURF);

      /* Position... */
      START_TIMER(TIMER_POSITION);
#pragma omp parallel for default(shared) private(ip)
      for (ip = 0; ip < atm->np; ip++)
        module_position(met0, met1, atm, ip);
      STOP_TIMER(TIMER_POSITION);

      /* Meteorological data... */
      START_TIMER(TIMER_METEO);
      if (ctl.met_dt_out > 0
          && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0)) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          module_meteo(&ctl, met0, met1, atm, ip);
      }
      STOP_TIMER(TIMER_METEO);

      /* Decay... */
      START_TIMER(TIMER_DECAY);
      if ((ctl.tdec_trop > 0 || ctl.tdec_strat > 0) && ctl.qnt_m >= 0) {
#pragma omp parallel for default(shared) private(ip)
        for (ip = 0; ip < atm->np; ip++)
          if (gsl_finite(dt[ip]))
            module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
      }
      STOP_TIMER(TIMER_DECAY);

      /* Write output... */
      START_TIMER(TIMER_OUTPUT);
      write_output(dirname, &ctl, met0, met1, atm, t);
      STOP_TIMER(TIMER_OUTPUT);
    }

    /* ------------------------------------------------------------
       Finalize model run...
       ------------------------------------------------------------ */

    /* Report memory usage... */
    printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
    printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
    printf("MEMORY_DYNAMIC = %g MByte\n",
           4 * NP * sizeof(double) / 1024. / 1024.);
    printf("MEMORY_STATIC = %g MByte\n",
           ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
            + (EX * EY + EX * EY * EP) * sizeof(float)
            + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);

    /* Report problem size... */
    printf("SIZE_NP = %d\n", atm->np);
    printf("SIZE_TASKS = %d\n", size);
    printf("SIZE_THREADS = %d\n", omp_get_max_threads());

    /* Report timers... */
    STOP_TIMER(TIMER_TOTAL);
    PRINT_TIMER(TIMER_TOTAL);
    PRINT_TIMER(TIMER_INIT);
    PRINT_TIMER(TIMER_INPUT);
    PRINT_TIMER(TIMER_OUTPUT);
    PRINT_TIMER(TIMER_ADVECT);
    PRINT_TIMER(TIMER_DECAY);
    PRINT_TIMER(TIMER_DIFFMESO);
    PRINT_TIMER(TIMER_DIFFTURB);
    PRINT_TIMER(TIMER_ISOSURF);
    PRINT_TIMER(TIMER_METEO);
    PRINT_TIMER(TIMER_POSITION);
    PRINT_TIMER(TIMER_SEDI);

    /* Free random number generators... */
    for (i = 0; i < NTHREADS; i++)
      gsl_rng_free(rng[i]);

    /* Free... */
    free(atm);
    free(met0);
    free(met1);
    free(dt);
  }

#ifdef MPI
  /* Finalize MPI... */
  MPI_Finalize();
#endif
```

```

  return EXIT_SUCCESS;
}

/*****************************************************************************/

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

  double v[3], xm[3];

  /* Interpolate meteorological data... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                  atm->lon[ip], atm->lat[ip], NULL, NULL, NULL, NULL,
                  &v[0], &v[1], &v[2], NULL, NULL, NULL);

  /* Get position of the mid point... */
  xm[0] = atm->lon[ip] + DX2DEG(0.5 * dt * v[0] / 1000., atm->lat[ip]);
  xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt * v[1] / 1000.);
  xm[2] = atm->p[ip] + 0.5 * dt * v[2];

  /* Interpolate meteorological data for mid point... */
  intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
                  xm[2], xm[0], xm[1], NULL, NULL, NULL, NULL,
                  &v[0], &v[1], &v[2], NULL, NULL, NULL);

  /* Save new position... */
  atm->time[ip] += dt;
  atm->lon[ip] += DX2DEG(dt * v[0] / 1000., xm[1]);
  atm->lat[ip] += DY2DEG(dt * v[1] / 1000.);
  atm->p[ip] += dt * v[2];
}

/*****************************************************************************/

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

  double ps, pt, tdec;

  /* Set constant lifetime... */
  if (ctl->tdec_trop == ctl->tdec_strat)
    tdec = ctl->tdec_trop;

  /* Set altitude-dependent lifetime... */
  else {

    /* Get surface pressure... */
    intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                    atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
                    NULL, NULL, NULL, NULL, NULL, NULL);

    /* Get tropopause pressure... */
    pt = clim_tropo(atm->time[ip], atm->lat[ip]);

    /* Set lifetime... */
    if (atm->p[ip] <= pt)
      tdec = ctl->tdec_strat;
    else
      tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
p[ip]);
  }

  /* Calculate exponential decay... */
  atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
}

/*****************************************************************************/

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

```

```
  double r, rs, u[16], v[16], w[16];

  int ix, iy, iz;

  /* Get indices... */
  ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
  iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
  iz = locate_irr(met0->p, met0->np, atm->p[ip]);

  /* Caching of wind standard deviations... */
  if (atm->cache_time[ix][iy][iz] != met0->time) {

    /* Collect local wind data... */
    u[0] = met0->u[ix][iy][iz];
    u[1] = met0->u[ix + 1][iy][iz];
    u[2] = met0->u[ix][iy + 1][iz];
    u[3] = met0->u[ix + 1][iy + 1][iz];
    u[4] = met0->u[ix][iy][iz + 1];
    u[5] = met0->u[ix + 1][iy][iz + 1];
    u[6] = met0->u[ix][iy + 1][iz + 1];
    u[7] = met0->u[ix + 1][iy + 1][iz + 1];

    v[0] = met0->v[ix][iy][iz];
    v[1] = met0->v[ix + 1][iy][iz];
    v[2] = met0->v[ix][iy + 1][iz];
    v[3] = met0->v[ix + 1][iy + 1][iz];
    v[4] = met0->v[ix][iy][iz + 1];
    v[5] = met0->v[ix + 1][iy][iz + 1];
    v[6] = met0->v[ix][iy + 1][iz + 1];
    v[7] = met0->v[ix + 1][iy + 1][iz + 1];

    w[0] = met0->w[ix][iy][iz];
    w[1] = met0->w[ix + 1][iy][iz];
    w[2] = met0->w[ix][iy + 1][iz];
    w[3] = met0->w[ix + 1][iy + 1][iz];
    w[4] = met0->w[ix][iy][iz + 1];
    w[5] = met0->w[ix + 1][iy][iz + 1];
    w[6] = met0->w[ix][iy + 1][iz + 1];
    w[7] = met0->w[ix + 1][iy + 1][iz + 1];

    /* Collect local wind data... */
    u[8] = met1->u[ix][iy][iz];
    u[9] = met1->u[ix + 1][iy][iz];
    u[10] = met1->u[ix][iy + 1][iz];
    u[11] = met1->u[ix + 1][iy + 1][iz];
    u[12] = met1->u[ix][iy][iz + 1];
    u[13] = met1->u[ix + 1][iy][iz + 1];
    u[14] = met1->u[ix][iy + 1][iz + 1];
    u[15] = met1->u[ix + 1][iy + 1][iz + 1];

    v[8] = met1->v[ix][iy][iz];
    v[9] = met1->v[ix + 1][iy][iz];
    v[10] = met1->v[ix][iy + 1][iz];
    v[11] = met1->v[ix + 1][iy + 1][iz];
    v[12] = met1->v[ix][iy][iz + 1];
    v[13] = met1->v[ix + 1][iy][iz + 1];
    v[14] = met1->v[ix][iy + 1][iz + 1];
    v[15] = met1->v[ix + 1][iy + 1][iz + 1];

    w[8] = met1->w[ix][iy][iz];
    w[9] = met1->w[ix + 1][iy][iz];
    w[10] = met1->w[ix][iy + 1][iz];
    w[11] = met1->w[ix + 1][iy + 1][iz];
    w[12] = met1->w[ix][iy][iz + 1];
    w[13] = met1->w[ix + 1][iy][iz + 1];
    w[14] = met1->w[ix][iy + 1][iz + 1];
    w[15] = met1->w[ix + 1][iy + 1][iz + 1];

    /* Get standard deviations of local wind data... */
    atm->cache_usig[ix][iy][iz] = (float) gsl_stats_sd(u, 1, 16);
    atm->cache_vsig[ix][iy][iz] = (float) gsl_stats_sd(v, 1, 16);
    atm->cache_wsig[ix][iy][iz] = (float) gsl_stats_sd(w, 1, 16);
    atm->cache_time[ix][iy][iz] = met0->time;
  }

  /* Set temporal correlations for mesoscale fluctuations... */
  r = 1 - 2 * fabs(dt) / ctl->dt_met;
  rs = sqrt(1 - r * r);

  /* Calculate horizontal mesoscale wind fluctuations... */
  if (ctl->turb_mesox > 0) {
    atm->up[ip] = (float)
      (r * atm->up[ip]
       + rs * gsl_ran_gaussian_ziggurat(rng,
                                        ctl->turb_mesox *
                                        atm->cache_usig[ix][iy][iz]));
    atm->lon[ip] += DX2DEG(atm->up[ip] * dt / 1000., atm->lat[ip]);
    atm->vp[ip] = (float)
      (r * atm->vp[ip]
       + rs * gsl_ran_gaussian_ziggurat(rng,
                                        ctl->turb_mesox *
                                        atm->cache_vsig[ix][iy][iz]));
    atm->lat[ip] += DY2DEG(atm->vp[ip] * dt / 1000.);
  }

  /* Calculate vertical mesoscale wind fluctuations... */
  if (ctl->turb_mesoz > 0) {
    atm->wp[ip] = (float)
      (r * atm->wp[ip]
       + rs * gsl_ran_gaussian_ziggurat(rng,
                                        ctl->turb_mesoz *
                                        atm->cache_wsig[ix][iy][iz]));
    atm->p[ip] += atm->wp[ip] * dt;
  }
}

/*****************************************************************************/

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

  double dx, dz, pt, p0, p1, w;

  /* Get tropopause pressure... */
  pt = clim_tropo(atm->time[ip], atm->lat[ip]);

  /* Get weighting factor... */
  p1 = pt * 0.866877899;
  p0 = pt / 0.866877899;
  if (atm->p[ip] > p0)
    w = 1;
  else if (atm->p[ip] < p1)
    w = 0;
  else
    w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);

  /* Set diffusivity... */
  dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
  dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;

  /* Horizontal turbulent diffusion... */
  if (dx > 0) {
    atm->lon[ip]
      += DX2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
                / 1000., atm->lat[ip]);
    atm->lat[ip]
      += DY2DEG(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
                / 1000.);
  }

  /* Vertical turbulent diffusion... */
  if (dz > 0)
    atm->p[ip]
      += DZ2DP(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
               / 1000., atm->p[ip]);
}

/*****************************************************************************/

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

  static double *iso, *ps, t, *ts;

  static int idx, ip2, n;

  FILE *in;

  char line[LEN];

  /* Initialize... */
  if (ip < 0) {

    /* Allocate... */
    ALLOC(iso, double,
          NP);
    ALLOC(ps, double,
          NP);
    ALLOC(ts, double,
          NP);

    /* Save pressure... */
    if (ctl->isosurf == 1)
      for (ip2 = 0; ip2 < atm->np; ip2++)
        iso[ip2] = atm->p[ip2];

    /* Save density... */
    else if (ctl->isosurf == 2)
      for (ip2 = 0; ip2 < atm->np; ip2++) {
        intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                        atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
                        &t, NULL, NULL, NULL, NULL, NULL, NULL);
        iso[ip2] = atm->p[ip2] / t;
      }

    /* Save potential temperature... */
    else if (ctl->isosurf == 3)
      for (ip2 = 0; ip2 < atm->np; ip2++) {
        intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
                        atm->lon[ip2], atm->lat[ip2], NULL, NULL, NULL,
                        &t, NULL, NULL, NULL, NULL, NULL, NULL);
        iso[ip2] = THETA(atm->p[ip2], t);
      }

    /* Read balloon pressure data... */
    else if (ctl->isosurf == 4) {

      /* Write info... */
      printf("Read balloon pressure data: %s\n", ctl->balloon);

      /* Open file... */
      if (!(in = fopen(ctl->balloon, "r")))
        ERRMSG("Cannot open file!");

      /* Read pressure time series... */
      while (fgets(line, LEN, in))
        if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
          if ((++n) > NP)
            ERRMSG("Too many data points!");

      /* Check number of points... */
      if (n < 1)
        ERRMSG("Could not read any data!");

      /* Close file... */
      fclose(in);
    }

    /* Leave initialization... */
    return;
  }

  /* Restore pressure... */
  if (ctl->isosurf == 1)
    atm->p[ip] = iso[ip];

  /* Restore density... */
  else if (ctl->isosurf == 2) {
    intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
                    atm->lat[ip], NULL, NULL, NULL, &t,
                    NULL, NULL, NULL, NULL, NULL, NULL);
    atm->p[ip] = iso[ip] * t;
  }

  /* Restore potential temperature... */
  else if (ctl->isosurf == 3) {
    intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
                    atm->lat[ip], NULL, NULL, NULL, &t,
                    NULL, NULL, NULL, NULL, NULL, NULL);
    atm->p[ip] = 1000. * pow(iso[ip] / t, -1. / 0.286);
  }

  /* Interpolate pressure... */
  else if (ctl->isosurf == 4) {
    if (atm->time[ip] <= ts[0])
      atm->p[ip] = ps[0];
    else if (atm->time[ip] >= ts[n - 1])
      atm->p[ip] = ps[n - 1];
    else {
      idx = locate_irr(ts, n, atm->time[ip]);
      atm->p[ip] = LIN(ts[idx], ps[idx],
                       ts[idx + 1], ps[idx + 1], atm->time[ip]);
    }
  }
}

/*****************************************************************************/

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

  double a, b, c, ps, pt, pv, p_hno3, p_h2o, t, u, v, w, x1, x2, h2o, o3, z;

  /* Interpolate meteorological data... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
lon[ip],
                  atm->lat[ip], &ps, &pt, &z, &t, &u, &v, &w, &pv, &h2o, &o3);

  /* Set surface pressure... */
  if (ctl->qnt_ps >= 0)
    atm->q[ctl->qnt_ps][ip] = ps;

  /* Set tropopause pressure... */
  if (ctl->qnt_pt >= 0)
    atm->q[ctl->qnt_pt][ip] = pt;

  /* Set pressure... */
  if (ctl->qnt_p >= 0)
    atm->q[ctl->qnt_p][ip] = atm->p[ip];

  /* Set geopotential height... */
  if (ctl->qnt_z >= 0)
    atm->q[ctl->qnt_z][ip] = z;

  /* Set temperature... */
  if (ctl->qnt_t >= 0)
    atm->q[ctl->qnt_t][ip] = t;

  /* Set zonal wind... */
  if (ctl->qnt_u >= 0)
    atm->q[ctl->qnt_u][ip] = u;

  /* Set meridional wind... */
  if (ctl->qnt_v >= 0)
    atm->q[ctl->qnt_v][ip] = v;

  /* Set vertical velocity... */
  if (ctl->qnt_w >= 0)
    atm->q[ctl->qnt_w][ip] = w;

  /* Set water vapor vmr... */
  if (ctl->qnt_h2o >= 0)
    atm->q[ctl->qnt_h2o][ip] = h2o;

  /* Set ozone vmr... */
  if (ctl->qnt_o3 >= 0)
    atm->q[ctl->qnt_o3][ip] = o3;

  /* Calculate horizontal wind... */
  if (ctl->qnt_vh >= 0)
    atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);

  /* Calculate vertical velocity... */
  if (ctl->qnt_vz >= 0)
    atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;

  /* Calculate potential temperature... */
  if (ctl->qnt_theta >= 0)
    atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);

  /* Set potential vorticity... */
  if (ctl->qnt_pv >= 0)
    atm->q[ctl->qnt_pv][ip] = pv;

  /* Calculate T_ice (Marti and Mauersberger, 1993)... */
  if (ctl->qnt_tice >= 0)
    atm->q[ctl->qnt_tice][ip] =
      -2663.5 /
      (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
       12.537);

  /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
  if (ctl->qnt_tnat >= 0) {
    if (ctl->psc_hno3 > 0)
      p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
    else
      p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
        * 1e-9 * atm->p[ip] / 1.333224;
    p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
    a = 0.009179 - 0.00088 * log10(p_h2o);
    b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
    c = -11397.0 / a;
    x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
    x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
    if (x1 > 0)
      atm->q[ctl->qnt_tnat][ip] = x1;
    if (x2 > 0)
      atm->q[ctl->qnt_tnat][ip] = x2;
  }

  /* Calculate T_STS (mean of T_ice and T_NAT)... */
  if (ctl->qnt_tsts >= 0) {
    if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
      ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
    atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
                                       + atm->q[ctl->qnt_tnat][ip]);
  }
}

/*****************************************************************************/

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

  double ps;

  /* Calculate modulo... */
  atm->lon[ip] = fmod(atm->lon[ip], 360);
  atm->lat[ip] = fmod(atm->lat[ip], 360);

  /* Check latitude... */
  while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
    if (atm->lat[ip] > 90) {
      atm->lat[ip] = 180 - atm->lat[ip];
      atm->lon[ip] += 180;
    }
    if (atm->lat[ip] < -90) {
      atm->lat[ip] = -180 - atm->lat[ip];
      atm->lon[ip] += 180;
    }
  }

  /* Check longitude... */
  while (atm->lon[ip] < -180)
    atm->lon[ip] += 360;
  while (atm->lon[ip] >= 180)
    atm->lon[ip] -= 360;

  /* Get surface pressure... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
                  atm->lon[ip], atm->lat[ip], &ps, NULL, NULL, NULL,
                  NULL, NULL, NULL, NULL, NULL, NULL);

  /* Check pressure... */
  if (atm->p[ip] > ps)
    atm->p[ip] = ps;
  else if (atm->p[ip] < met0->p[met0->np - 1])
    atm->p[ip] = met0->p[met0->np - 1];
}

/*****************************************************************************/

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

  /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
  const double A = 1.249, B = 0.42, C = 0.87;

  /* Average mass of an air molecule [kg/molec]: */
  const double m = 4.8096e-26;

  double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;

  /* Convert units... */
  p = 100 * atm->p[ip];
  r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
  rho_p = atm->q[ctl->qnt_rho][ip];

  /* Get temperature... */
  intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->lon[ip],
                  atm->lat[ip], NULL, NULL, NULL, &T,
                  NULL, NULL, NULL, NULL, NULL, NULL);

  /* Density of dry air... */
  rho = p / (RA * T);

  /* Dynamic viscosity of air... */
  eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Thermal velocity of an air molecule... */
  v = sqrt(8 * KB * T / (M_PI * m));

  /* Mean free path of an air molecule... */
  lambda = 2 * eta / (rho * v);

  /* Knudsen number for air... */
  K = lambda / r_p;

  /* Cunningham slip-flow correction... */
  G = 1 + K * (A + B * exp(-C / K));

  /* Sedimentation (fall) velocity... */
  v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;

  /* Calculate pressure change... */
  atm->p[ip] += DZ2DP(v_p * dt / 1000., atm->p[ip]);
}

/*****************************************************************************/

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

  char filename[2 * LEN];

  double r;

  int year, mon, day, hour, min, sec;

  /* Get time... */
  jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Write atmospheric data... */
  if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
            dirname, ctl->atm_basename, year, mon, day, hour, min);
    write_atm(filename, ctl, atm, t);
  }

  /* Write CSI data... */
  if (ctl->csi_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
    write_csi(filename, ctl, atm, t);
  }

  /* Write ensemble data... */
  if (ctl->ens_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
    write_ens(filename, ctl, atm, t);
  }

  /* Write gridded data... */
  if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
            dirname, ctl->grid_basename, year, mon, day, hour, min);
    write_grid(filename, ctl, met0, met1, atm, t);
  }

  /* Write profile data... */
  if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write_prof(filename, ctl, met0, met1, atm, t);
  }

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

Index