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).

4.1.1 Detailed Description

Atmospheric data.

Definition at line 718 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

Definition at line 721 of file libtrac.h.

4.1.2.2 double atm_t::time[NP]

Time [s].

Definition at line 724 of file libtrac.h.

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

Definition at line 727 of file libtrac.h.

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 730 of file libtrac.h.

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 733 of file libtrac.h.

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

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

Definition at line 736 of file libtrac.h.

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

- libtrac.h

4.2 cache_t Struct Reference

Cache data.

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

Data Fields

- 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 iso_var [NP]
  Isosurface variables.
- double iso_ps [NP]
  Isosurface balloon pressure [hPa].
- double iso_ts [NP]
  Isosurface balloon time [s].
- int iso_n
  Isosurface balloon number of data points.
- double tsig [EX][EY][EP]
  Cache for reference time of wind standard deviations.
- float usig [EX][EY][EP]
  Cache for zonal wind standard deviations.
- float vsig [EX][EY][EP]
  Cache for meridional wind standard deviations.
- float wsig [EX][EY][EP]
  Cache for vertical velocity standard deviations.

4.2.1 Detailed Description

Cache data.

Definition at line 741 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 float cache_t::up[NP]

Zonal wind perturbation [m/s].

Definition at line 744 of file libtrac.h.

4.2.2.2 float cache_t::vp[NP]

Meridional wind perturbation [m/s].

Definition at line 747 of file libtrac.h.

4.2.2.3 float cache_t::wp[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 750 of file libtrac.h.

4.2.2.4 double cache_t::iso_var[NP]

Isosurface variables.

Definition at line 753 of file libtrac.h.

4.2.2.5 double cache_t::iso_ps[NP]

Isosurface balloon pressure [hPa].

Definition at line 756 of file libtrac.h.

4.2.2.6 double cache_t::iso_ts[NP]

Isosurface balloon time [s].

Definition at line 759 of file libtrac.h.

4.2.2.7 int cache_t::iso_n

Isosurface balloon number of data points.

Definition at line 762 of file libtrac.h.

4.2.2.8 double cache_t::tsig[EX][EY][EP]

Cache for reference time of wind standard deviations.

Definition at line 765 of file libtrac.h.

4.2.2.9 float cache_t::usig[EX][EY][EP]

Cache for zonal wind standard deviations.

Definition at line 768 of file libtrac.h.

4.2.2.10 float cache_t::vsig[EX][EY][EP]

Cache for meridional wind standard deviations.

Definition at line 771 of file libtrac.h.

4.2.2.11 float cache_t::wsig[EX][EY][EP]

Cache for vertical velocity standard deviations.

Definition at line 774 of file libtrac.h.

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

- libtrac.h

4.3 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_lwc
  Quantity array index for cloud liquid water content.
- int qnt_iwc
  Quantity array index for cloud ice water content.
- int qnt_pc
  Quantity array index for cloud top pressure.

- int qnt_hno3

  Quantity array index for nitric acid vmr.
- int qnt_oh

  Quantity array index for hydroxyl number concentrations.
- int qnt_rh

  Quantity array index for relative humidty.
- 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.
- char species [LEN]

  Species.
- 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 oh_chem [4]

  Coefficients for OH chemistry (k0, n, kinf, m).
- double wet_depo [4]

  Coefficients for wet deposition (A, B, H).
- 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_stride

  Particle index stride for atmospheric data files.
- 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].

4.3.1 Detailed Description

Control parameters.

Definition at line 369 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 int ctl_t::nq

Number of quantities.

Definition at line 372 of file libtrac.h.

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

Quantity names.

Definition at line 375 of file libtrac.h.

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

Quantity units.

Definition at line 378 of file libtrac.h.

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

Quantity output format.

Definition at line 381 of file libtrac.h.

4.3.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 384 of file libtrac.h.

4.3.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 387 of file libtrac.h.

4.3.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 390 of file libtrac.h.

4.3.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 393 of file libtrac.h.

4.3.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 396 of file libtrac.h.

4.3.2.10 int ctl_t::qnt_pt

Quantity array index for tropopause pressure.

Definition at line 399 of file libtrac.h.

4.3.2.11 int ctl_t::qnt_z

Quantity array index for geopotential height.

Definition at line 402 of file libtrac.h.

4.3.2.12 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 405 of file libtrac.h.

4.3.2.13 int ctl_t::qnt_t

Quantity array index for temperature.

Definition at line 408 of file libtrac.h.

4.3.2.14 int ctl_t::qnt_u

Quantity array index for zonal wind.

Definition at line 411 of file libtrac.h.

4.3.2.15 int ctl_t::qnt_v

Quantity array index for meridional wind.

Definition at line 414 of file libtrac.h.

4.3.2.16 int ctl_t::qnt_w

Quantity array index for vertical velocity.

Definition at line 417 of file libtrac.h.

4.3.2.17 int ctl_t::qnt_h2o

Quantity array index for water vapor vmr.

Definition at line 420 of file libtrac.h.

4.3.2.18 int ctl_t::qnt_o3

Quantity array index for ozone vmr.

Definition at line 423 of file libtrac.h.

4.3.2.19 int ctl_t::qnt_lwc

Quantity array index for cloud liquid water content.

Definition at line 426 of file libtrac.h.

4.3.2.20 int ctl_t::qnt_iwc

Quantity array index for cloud ice water content.

Definition at line 429 of file libtrac.h.

4.3.2.21 int ctl_t::qnt_pc

Quantity array index for cloud top pressure.

Definition at line 432 of file libtrac.h.

4.3.2.22 int ctl_t::qnt_hno3

Quantity array index for nitric acid vmr.

Definition at line 435 of file libtrac.h.

4.3.2.23 int ctl_t::qnt_oh

Quantity array index for hydroxyl number concentrations.

Definition at line 438 of file libtrac.h.

4.3.2.24 int ctl_t::qnt_rh

Quantity array index for relative humidty.

Definition at line 441 of file libtrac.h.

4.3.2.25 int ctl_t::qnt_theta

Quantity array index for potential temperature.

Definition at line 444 of file libtrac.h.

4.3.2.26 int ctl_t::qnt_vh

Quantity array index for horizontal wind.

Definition at line 447 of file libtrac.h.

4.3.2.27 int ctl_t::qnt_vz

Quantity array index for vertical velocity.

Definition at line 450 of file libtrac.h.

4.3.2.28 int ctl_t::qnt_pv

Quantity array index for potential vorticity.

Definition at line 453 of file libtrac.h.

4.3.2.29 int ctl_t::qnt_tice

Quantity array index for T_ice.

Definition at line 456 of file libtrac.h.

4.3.2.30 int ctl_t::qnt_tsts

Quantity array index for T_STS.

Definition at line 459 of file libtrac.h.

4.3.2.31 int ctl_t::qnt_tnat

Quantity array index for T_NAT.

Definition at line 462 of file libtrac.h.

4.3.2.32 int ctl_t::qnt_stat

Quantity array index for station flag.

Definition at line 465 of file libtrac.h.

4.3.2.33 int ctl_t::direction

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

Definition at line 468 of file libtrac.h.

4.3.2.34 double ctl_t::t_start

Start time of simulation [s].

Definition at line 471 of file libtrac.h.

4.3.2.35 double ctl_t::t_stop

Stop time of simulation [s].

Definition at line 474 of file libtrac.h.

4.3.2.36 double ctl_t::dt_mod

Time step of simulation [s].

Definition at line 477 of file libtrac.h.

4.3.2.37 double ctl_t::dt_met

Time step of meteorological data [s].

Definition at line 480 of file libtrac.h.

4.3.2.38 int ctl_t::met_dx

Stride for longitudes.

Definition at line 483 of file libtrac.h.

4.3.2.39 int ctl_t::met_dy

Stride for latitudes.

Definition at line 486 of file libtrac.h.

4.3.2.40 int ctl_t::met_dp

Stride for pressure levels.

Definition at line 489 of file libtrac.h.

4.3.2.41 int ctl_t::met_sx

Smoothing for longitudes.

Definition at line 492 of file libtrac.h.

4.3.2.42 int ctl_t::met_sy

Smoothing for latitudes.

Definition at line 495 of file libtrac.h.

4.3.2.43 int ctl_t::met_sp

Smoothing for pressure levels.

Definition at line 498 of file libtrac.h.

4.3.2.44 int ctl_t::met_np

Number of target pressure levels.

Definition at line 501 of file libtrac.h.

4.3.2.45 double ctl_t::met_p[EP]

Target pressure levels [hPa].

Definition at line 504 of file libtrac.h.

4.3.2.46 int ctl_t::met_tropo

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

Definition at line 508 of file libtrac.h.

4.3.2.47 char ctl_t::met_geopot[LEN]

Surface geopotential data file.

Definition at line 511 of file libtrac.h.

4.3.2.48 double ctl_t::met_dt_out

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

Definition at line 514 of file libtrac.h.

4.3.2.49 char ctl_t::met_stage[LEN]

Command to stage meteo data.

Definition at line 517 of file libtrac.h.

4.3.2.50 int ctl_t::isosurf

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

Definition at line 521 of file libtrac.h.

4.3.2.51 char ctl_t::balloon[LEN]

Balloon position filename.

Definition at line 524 of file libtrac.h.

4.3.2.52 double ctl_t::turb_dx_trop

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

Definition at line 527 of file libtrac.h.

4.3.2.53 double ctl_t::turb_dx_strat

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

Definition at line 530 of file libtrac.h.

4.3.2.54 double ctl_t::turb_dz_trop

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

Definition at line 533 of file libtrac.h.

4.3.2.55 double ctl_t::turb_dz_strat

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

Definition at line 536 of file libtrac.h.

4.3.2.56 double ctl_t::turb_mesox

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 539 of file libtrac.h.

4.3.2.57 double ctl_t::turb_mesoz

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 542 of file libtrac.h.

4.3.2.58 char ctl_t::species[LEN]

Species.

Definition at line 545 of file libtrac.h.

4.3.2.59 double ctl_t::molmass

Molar mass [g/mol].

Definition at line 548 of file libtrac.h.

4.3.2.60 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 551 of file libtrac.h.

4.3.2.61 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 554 of file libtrac.h.

4.3.2.62 double ctl_t::oh_chem[4]

Coefficients for OH chemistry (k0, n, kinf, m).

Definition at line 557 of file libtrac.h.

4.3.2.63 double ctl_t::wet_depo[4]

Coefficients for wet deposition (A, B, H).

Definition at line 560 of file libtrac.h.

4.3.2.64 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 563 of file libtrac.h.

4.3.2.65 double ctl_t::psc_hno3

HNO3 volume mixing ratio for PSC analysis.

Definition at line 566 of file libtrac.h.

4.3.2.66 char ctl_t::atm_basename[LEN]

Basename of atmospheric data files.

Definition at line 569 of file libtrac.h.

4.3.2.67 char ctl_t::atm_gpfile[LEN]

Gnuplot file for atmospheric data.

Definition at line 572 of file libtrac.h.

4.3.2.68 double ctl_t::atm_dt_out

Time step for atmospheric data output [s].

Definition at line 575 of file libtrac.h.

4.3.2.69 int ctl_t::atm_filter

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

Definition at line 578 of file libtrac.h.

4.3.2.70 int ctl_t::atm_stride

Particle index stride for atmospheric data files.

Definition at line 581 of file libtrac.h.

4.3.2.71 int ctl_t::atm_type

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

Definition at line 584 of file libtrac.h.

4.3.2.72 char ctl_t::csi_basename[LEN]

Basename of CSI data files.

Definition at line 587 of file libtrac.h.

4.3.2.73 double ctl_t::csi_dt_out

Time step for CSI data output [s].

Definition at line 590 of file libtrac.h.

4.3.2.74 char ctl_t::csi_obsfile[LEN]

Observation data file for CSI analysis.

Definition at line 593 of file libtrac.h.

4.3.2.75 double ctl_t::csi_obsmin

Minimum observation index to trigger detection.

Definition at line 596 of file libtrac.h.

4.3.2.76 double ctl_t::csi_modmin

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

Definition at line 599 of file libtrac.h.

4.3.2.77 int ctl_t::csi_nz

Number of altitudes of gridded CSI data.

Definition at line 602 of file libtrac.h.

4.3.2.78 double ctl_t::csi_z0

Lower altitude of gridded CSI data [km].

Definition at line 605 of file libtrac.h.

4.3.2.79 double ctl_t::csi_z1

Upper altitude of gridded CSI data [km].

Definition at line 608 of file libtrac.h.

4.3.2.80 int ctl_t::csi_nx

Number of longitudes of gridded CSI data.

Definition at line 611 of file libtrac.h.

4.3.2.81 double ctl_t::csi_lon0

Lower longitude of gridded CSI data [deg].

Definition at line 614 of file libtrac.h.

4.3.2.82 double ctl_t::csi_lon1

Upper longitude of gridded CSI data [deg].

Definition at line 617 of file libtrac.h.

4.3.2.83 int ctl_t::csi_ny

Number of latitudes of gridded CSI data.

Definition at line 620 of file libtrac.h.

4.3.2.84 double ctl_t::csi_lat0

Lower latitude of gridded CSI data [deg].

Definition at line 623 of file libtrac.h.

4.3.2.85 double ctl_t::csi_lat1

Upper latitude of gridded CSI data [deg].

Definition at line 626 of file libtrac.h.

4.3.2.86 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 629 of file libtrac.h.

4.3.2.87 char ctl_t::grid_gpfile[LEN]

Gnuplot file for gridded data.

Definition at line 632 of file libtrac.h.

4.3.2.88 double ctl_t::grid_dt_out

Time step for gridded data output [s].

Definition at line 635 of file libtrac.h.

4.3.2.89 int ctl_t::grid_sparse

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

Definition at line 638 of file libtrac.h.

4.3.2.90 int ctl_t::grid_nz

Number of altitudes of gridded data.

Definition at line 641 of file libtrac.h.

4.3.2.91 double ctl_t::grid_z0

Lower altitude of gridded data [km].

Definition at line 644 of file libtrac.h.

4.3.2.92 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 647 of file libtrac.h.

4.3.2.93 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 650 of file libtrac.h.

4.3.2.94 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 653 of file libtrac.h.

4.3.2.95 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 656 of file libtrac.h.

4.3.2.96 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 659 of file libtrac.h.

4.3.2.97 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 662 of file libtrac.h.

4.3.2.98 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 665 of file libtrac.h.

4.3.2.99 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 668 of file libtrac.h.

4.3.2.100 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 671 of file libtrac.h.

4.3.2.101 int ctl_t::prof_nz

Number of altitudes of gridded profile data.

Definition at line 674 of file libtrac.h.

4.3.2.102 double ctl_t::prof_z0

Lower altitude of gridded profile data [km].

Definition at line 677 of file libtrac.h.

4.3.2.103 double ctl_t::prof_z1

Upper altitude of gridded profile data [km].

Definition at line 680 of file libtrac.h.

4.3.2.104 int ctl_t::prof_nx

Number of longitudes of gridded profile data.

Definition at line 683 of file libtrac.h.

4.3.2.105 double ctl_t::prof_lon0

Lower longitude of gridded profile data [deg].

Definition at line 686 of file libtrac.h.

4.3.2.106 double ctl_t::prof_lon1

Upper longitude of gridded profile data [deg].

Definition at line 689 of file libtrac.h.

4.3.2.107 int ctl_t::prof_ny

Number of latitudes of gridded profile data.

Definition at line 692 of file libtrac.h.

4.3.2.108 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 695 of file libtrac.h.

4.3.2.109 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 698 of file libtrac.h.

4.3.2.110 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 701 of file libtrac.h.

4.3.2.111 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 704 of file libtrac.h.

4.3.2.112 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 707 of file libtrac.h.

4.3.2.113 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 710 of file libtrac.h.

4.3.2.114 double ctl_t::stat_r

Search radius around station [km].

Definition at line 713 of file libtrac.h.

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

- libtrac.h

4.4 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].
- float ps [EX][EY]

  Surface pressure [hPa].
- float zs [EX][EY]

  Geopotential height at the surface [km].
- float pt [EX][EY]

  Tropopause pressure [hPa].
- float pc [EX][EY]

  Cloud top pressure [hPa].
- float cl [EX][EY]

  Total column cloud water [kg/m^2].
- float z [EX][EY][EP]

  Geopotential height at model levels [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 lwc [EX][EY][EP]

  Cloud liquid water content [kg/kg].
- float iwc [EX][EY][EP]

  Cloud ice water content [kg/kg].
- float pl [EX][EY][EP]

  Pressure on model levels [hPa].

4.4.1 Detailed Description

Meteorological data.

Definition at line 779 of file libtrac.h.

4.4.2 Field Documentation

4.4.2.1 double met_t::time

Time [s].

Definition at line 782 of file libtrac.h.

4.4.2.2 int met_t::nx

Number of longitudes.

Definition at line 785 of file libtrac.h.

4.4.2.3 int met_t::ny

Number of latitudes.

Definition at line 788 of file libtrac.h.

4.4.2.4 int met_t::np

Number of pressure levels.

Definition at line 791 of file libtrac.h.

4.4.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 794 of file libtrac.h.

4.4.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 797 of file libtrac.h.

4.4.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 800 of file libtrac.h.

4.4.2.8 float met_t::ps[EX][EY]

Surface pressure [hPa].

Definition at line 803 of file libtrac.h.

4.4.2.9 float met_t::zs[EX][EY]

Geopotential height at the surface [km].

Definition at line 806 of file libtrac.h.

4.4.2.10 float met_t::pt[EX][EY]

Tropopause pressure [hPa].

Definition at line 809 of file libtrac.h.

4.4.2.11 float met_t::pc[EX][EY]

Cloud top pressure [hPa].

Definition at line 812 of file libtrac.h.

4.4.2.12 float met_t::cl[EX][EY]

Total column cloud water [kg/m^2].

Definition at line 815 of file libtrac.h.

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

Geopotential height at model levels [km].

Definition at line 818 of file libtrac.h.

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

Temperature [K].

Definition at line 821 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 824 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 827 of file libtrac.h.

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

Vertical wind [hPa/s].

Definition at line 830 of file libtrac.h.

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

Potential vorticity [PVU].

Definition at line 833 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 836 of file libtrac.h.

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

Ozone volume mixing ratio [1].

Definition at line 839 of file libtrac.h.

4.4.2.21 float met_t::lwc[EX][EY][EP]

Cloud liquid water content [kg/kg].

Definition at line 842 of file libtrac.h.

4.4.2.22 float met_t::iwc[EX][EY][EP]

Cloud ice water content [kg/kg].

Definition at line 845 of file libtrac.h.

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

Pressure on model levels [hPa].

Definition at line 848 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.

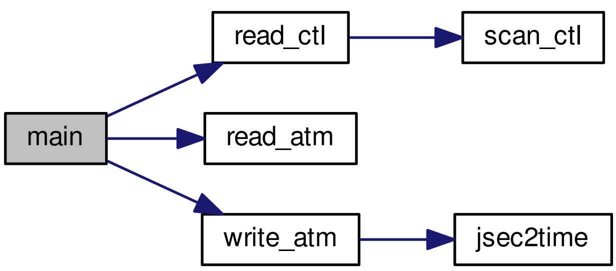
5.1.2 Function Documentation

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.

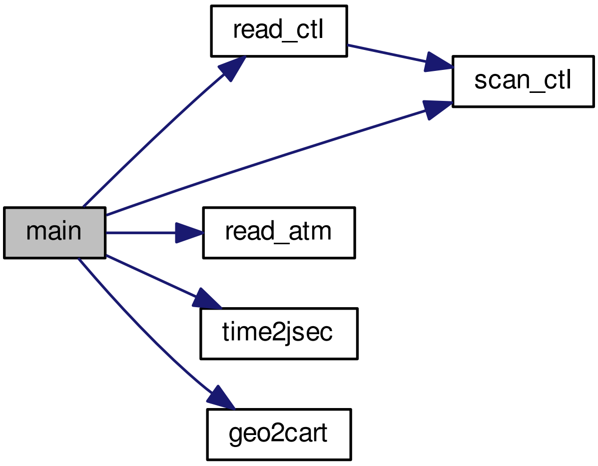
```
                {

  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]);
    }

    /* Get relative transport deviations... */
    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]);
    }

    /* Get relative transport deviations... */
    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

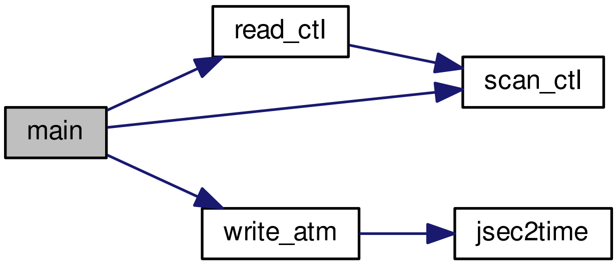
5.5.2.1 int main (int *argc*, char * *argv[ ]*)

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, "0", 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, 0);

  /* 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, "0", 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, 0);

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

  return EXIT_SUCCESS;
}
```

5.7 atm_select.c File Reference

Extract subsets of air parcels from atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.7.1 Detailed Description

Extract subsets of air parcels from atmospheric data files.

Definition in file atm_select.c.

5.7.2 Function Documentation

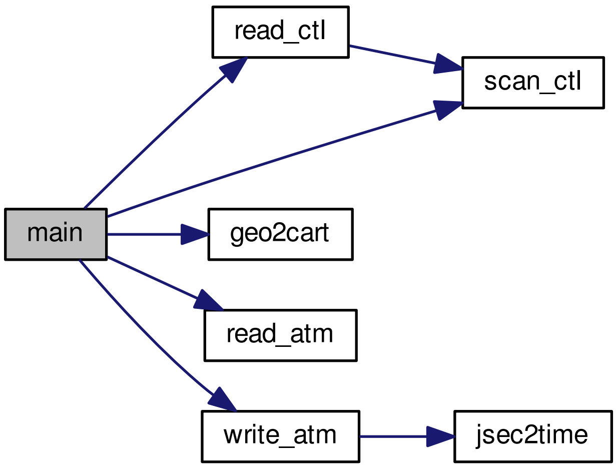
5.7.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 27 of file atm_select.c.

```
                 {

  ctl_t ctl;

  atm_t *atm, *atm2;

  double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
    x1[3];

  int f, ip, ip0, ip1, iq, stride;

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

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

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  stride =
    (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);
  ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "0", NULL);
  ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "0", NULL);
  t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
  t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
  p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
  p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
  lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
  r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
  r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
  rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
  rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);

  /* Get Cartesian coordinates... */
  geo2cart(0, rlon, rlat, x0);

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

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

```

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

      /* Check air parcel index... */
      if (ip0 != ip1)
        if ((ip0 < ip1 && (ip < ip0 || ip > ip1))
            || (ip0 > ip1 && (ip < ip0 && ip > ip1)))
          continue;

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

      /* Check vertical distance... */
      if (p0 != p1)
        if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
            || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
          continue;

      /* Check longitude... */
      if (lon0 != lon1)
        if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
            || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
          continue;

      /* Check latitude... */
      if (lat0 != lat1)
        if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
            || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
          continue;

      /* Check horizontal distace... */
      if (r0 != r1) {
        geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
        r = DIST(x0, x1);
        if ((r1 > r0 && (r < r0 || r > r1))
            || (r1 < r0 && (r < r0 && r > r1)))
          continue;
      }

      /* Copy data... */
      atm2->time[atm2->np] = atm->time[ip];
      atm2->p[atm2->np] = atm->p[ip];
      atm2->lon[atm2->np] = atm->lon[ip];
      atm2->lat[atm2->np] = atm->lat[ip];
      for (iq = 0; iq < ctl.nq; iq++)
        atm2->q[iq][atm2->np] = atm->q[iq][ip];
      if ((++atm2->np) > NP)
        ERRMSG("Too many air parcels!");
    }
  }

  /* Close file... */
  write_atm(argv[2], &ctl, atm2, 0);

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.8 atm_select.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) 2013-2020 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  atm_t *atm, *atm2;

  double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
    x1[3];

  int f, ip, ip0, ip1, iq, stride;

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

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

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

```
  stride =
    (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);
  ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "0", NULL);
  ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "0", NULL);
  t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
  t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
  p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
  p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
  lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
  r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
  r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
  rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
  rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);

  /* Get Cartesian coordinates... */
  geo2cart(0, rlon, rlat, x0);

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

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

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

      /* Check air parcel index... */
      if (ip0 != ip1)
        if ((ip0 < ip1 && (ip < ip0 || ip > ip1))
            || (ip0 > ip1 && (ip < ip0 && ip > ip1)))
          continue;

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

      /* Check vertical distance... */
      if (p0 != p1)
        if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
            || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
          continue;

      /* Check longitude... */
      if (lon0 != lon1)
        if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
            || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
          continue;

      /* Check latitude... */
      if (lat0 != lat1)
        if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
            || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
          continue;

      /* Check horizontal distace... */
      if (r0 != r1) {
        geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
        r = DIST(x0, x1);
        if ((r1 > r0 && (r < r0 || r > r1))
            || (r1 < r0 && (r < r0 && r > r1)))
          continue;
      }

      /* Copy data... */
      atm2->time[atm2->np] = atm->time[ip];
      atm2->p[atm2->np] = atm->p[ip];
      atm2->lon[atm2->np] = atm->lon[ip];
      atm2->lat[atm2->np] = atm->lat[ip];
      for (iq = 0; iq < ctl.nq; iq++)
        atm2->q[iq][atm2->np] = atm->q[iq][ip];
      if ((++atm2->np) > NP)
        ERRMSG("Too many air parcels!");
    }
  }

  /* Close file... */
  write_atm(argv[2], &ctl, atm2, 0);

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

```
  return EXIT_SUCCESS;
}
```

5.9 atm_split.c File Reference

Split air parcels into a larger number of parcels.

Functions

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

5.9.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file atm_split.c.

5.9.2 Function Documentation

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

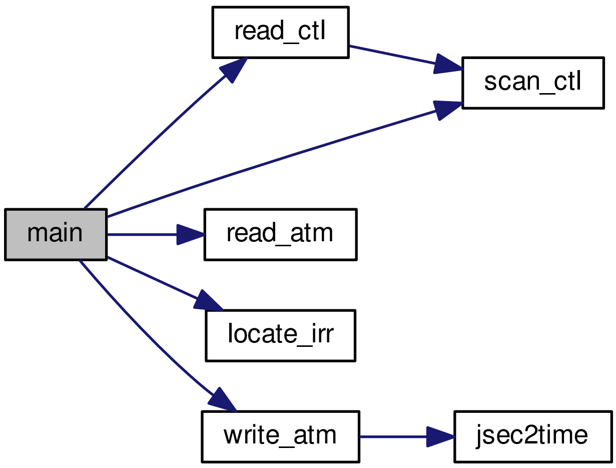
Definition at line 27 of file atm_split.c.

```
                    {

  atm_t *atm, *atm2;

  ctl_t ctl;

  gsl_rng *rng;

  FILE *in;

  char kernel[LEN], line[LEN];

  double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
    t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;

  int i, ip, iq, iz, n, nz = 0;

  /* 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);
  scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);
```

```

  /* 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!");

  /* Read kernel function... */
  if (kernel[0] != '-') {

    /* Write info... */
    printf("Read kernel function: %s\n", kernel);

    /* Open file... */
    if (!(in = fopen(kernel, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
        if ((++nz) >= GZ)
          ERRMSG("Too many height levels!");

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

    /* Normalize kernel function... */
    zmax = gsl_stats_max(kz, 1, (size_t) nz);
    zmin = gsl_stats_min(kz, 1, (size_t) nz);
    kmax = gsl_stats_max(kk, 1, (size_t) nz);
    kmin = gsl_stats_min(kk, 1, (size_t) nz);
    for (iz = 0; iz < nz; iz++)
      kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
  }

  /* 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 (nz > 0) {
      do {
        z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
        iz = locate_irr(kz, nz, z);
        k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
      } while (gsl_rng_uniform(rng) > k);
      atm2->p[atm2->np] = P(z);
    } else 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, 0);

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.10 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;

  FILE *in;

  char kernel[LEN], line[LEN];

  double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
    t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;

  int i, ip, iq, iz, n, nz = 0;

  /* 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);
  scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);

  /* 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!");

  /* Read kernel function... */
  if (kernel[0] != '-') {

    /* Write info... */
    printf("Read kernel function: %s\n", kernel);

    /* Open file... */
    if (!(in = fopen(kernel, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
        if ((++nz) >= GZ)
          ERRMSG("Too many height levels!");

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

    /* Normalize kernel function... */
    zmax = gsl_stats_max(kz, 1, (size_t) nz);
    zmin = gsl_stats_min(kz, 1, (size_t) nz);
    kmax = gsl_stats_max(kk, 1, (size_t) nz);
    kmin = gsl_stats_min(kk, 1, (size_t) nz);
    for (iz = 0; iz < nz; iz++)
      kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
  }

  /* 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 (nz > 0) {
      do {
        z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
        iz = locate_irr(kz, nz, z);
        k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
      } while (gsl_rng_uniform(rng) > k);
      atm2->p[atm2->np] = P(z);
    } else 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, 0);

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

  return EXIT_SUCCESS;
}
```

5.11 atm_stat.c File Reference

Calculate air parcel statistics.

Functions

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

5.11.1 Detailed Description

Calculate air parcel statistics.

Definition in file atm_stat.c.

5.11.2 Function Documentation

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

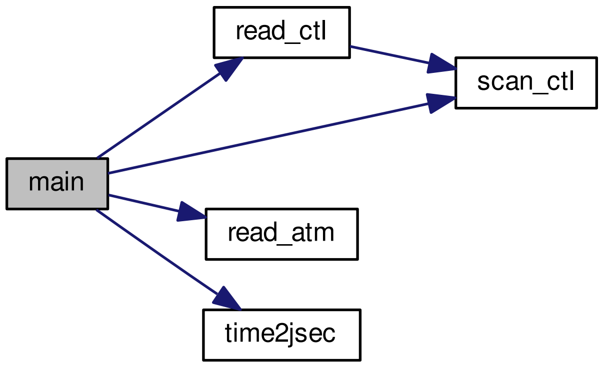
Definition at line 27 of file atm_stat.c.

```
                 {

  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:



5.12 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;
}
```

5.13 day2doy.c File Reference

Convert date to day of year.

Functions

- int main (int argc, char ∗argv[ ])

5.13.1 Detailed Description

Convert date to day of year.

Definition in file day2doy.c.

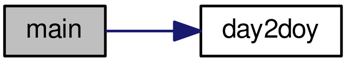
5.13.2 Function Documentation

5.13.2.1 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:



5.14 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;
}
```

5.15 doy2day.c File Reference

Convert day of year to date.

Functions

- int main (int argc, char ∗argv[ ])

5.15.1 Detailed Description

Convert day of year to date.

Definition in file doy2day.c.

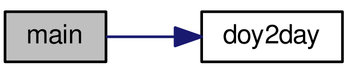
5.15.2 Function Documentation

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

Definition at line 27 of file doy2day.c.

```
                  {

  int day, doy, mon, year;

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

  /* 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;
}
```

Here is the call graph for this function:



5.16 doy2day.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 < 3)
    ERRMSG("Give parameters: <year> <doy>");

  /* 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;
}
```

5.17 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

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

5.17.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

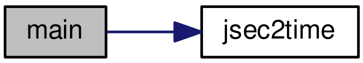
5.17.2 Function Documentation

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

Definition at line 27 of file jsec2time.c.

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

Here is the call graph for this function:



5.18 jsec2time.c

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

```
  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_oh (double t, double lat, double p)

  Climatology of OH number concentrations.
- 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_space_3d (met_t *met, float array[EX][EY][EP], double p, double lon, double lat, double *var, int *ci, double *cw, int init)

  Spatial interpolation of meteorological data.
- void intpol_met_space_2d (met_t *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)

  Spatial interpolation of meteorological data.
- void intpol_met_time_3d (met_t *met0, float array0[EX][EY][EP], met_t *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)

  Temporal interpolation of meteorological data.
- void intpol_met_time_2d (met_t *met0, float array0[EX][EY], met_t *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)

  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_cloud (met_t ∗met)

  Calculate cloud properties.
- void read_met_extrapolate (met_t ∗met)

  Extrapolate meteorological data at lower boundary.
- void read_met_geopot (met_t ∗met)

  Calculate geopotential heights.
- int read_met_help_3d (int ncid, char ∗varname, char ∗varname2, met_t ∗met, float dest[EX][EY][EP], float scl)

  Read and convert 3D variable from meteorological data file.
- int read_met_help_2d (int ncid, char ∗varname, char ∗varname2, met_t ∗met, float dest[EX][EY], float scl)

  Read and convert 2D 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_surface (int ncid, met_t ∗met)

  Read surface 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 spline (double ∗x, double ∗y, int n, double ∗x2, double ∗y2, int n2)

  Spline interpolation.
- double stddev (double ∗data, int n)

  Calculate standard deviation.
- 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 = 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*)

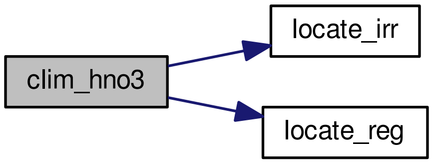
Climatology of HNO3 volume mixing ratios.

Definition at line 295 of file libtrac.c.

```
                {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Check pressure... */
  if (p < clim_hno3_ps[0])
    p = clim_hno3_ps[0];
  else if (p > clim_hno3_ps[9])
    p = clim_hno3_ps[9];

  /* Get indices... */
  int isec = locate_irr(clim_hno3_secs, 12, sec);
  int ilat = locate_reg(clim_hno3_lats, 18, lat);
  int ip = locate_irr(clim_hno3_ps, 10, p);

  /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
  double aux00 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec][ilat][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec][ilat + 1][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec + 1][ilat][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec + 1][ilat][ip + 1], p);
```

```
  double aux11 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec + 1][ilat + 1][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim_hno3_lats[ilat], aux00,
              clim_hno3_lats[ilat + 1], aux01, lat);
  aux11 = LIN(clim_hno3_lats[ilat], aux10,
              clim_hno3_lats[ilat + 1], aux11, lat);
  return LIN(clim_hno3_secs[isec], aux00,
             clim_hno3_secs[isec + 1], aux11, sec);
}
```

Here is the call graph for this function:



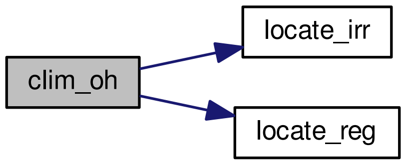
5.19.2.3 double clim_oh (double *t*, double *lat*, double *p*)

Climatology of OH number concentrations.

Definition at line 1322 of file libtrac.c.

```
                  {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Check pressure... */
  if (p < clim_oh_ps[0])
    p = clim_oh_ps[0];
  else if (p > clim_oh_ps[33])
    p = clim_oh_ps[33];

  /* Get indices... */
  int isec = locate_irr(clim_oh_secs, 12, sec);
  int ilat = locate_reg(clim_oh_lats, 18, lat);
  int ip = locate_irr(clim_oh_ps, 34, p);

  /* Interpolate OH climatology (Pommrich et al., 2014)... */
  double aux00 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec][ilat][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec][ilat + 1][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec + 1][ilat][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec + 1][ilat + 1][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
  aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
  return 1e6 * LIN(clim_oh_secs[isec], aux00,
                   clim_oh_secs[isec + 1], aux11, sec);
}
```

Here is the call graph for this function:



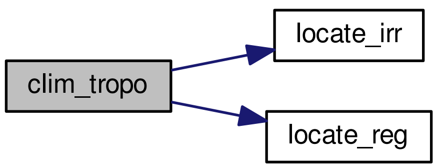
5.19.2.4 double clim_tropo (double *t*, double *lat*)

Climatology of tropopause pressure.

Definition at line 1497 of file libtrac.c.

```
                {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Get indices... */
  int isec = locate_irr(clim_tropo_secs, 12, sec);
  int ilat = locate_reg(clim_tropo_lats, 73, lat);

  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
  double p0 = LIN(clim_tropo_lats[ilat],
                  clim_tropo_tps[isec][ilat],
                  clim_tropo_lats[ilat + 1],
                  clim_tropo_tps[isec][ilat + 1], lat);
  double p1 = LIN(clim_tropo_lats[ilat],
                  clim_tropo_tps[isec + 1][ilat],
                  clim_tropo_lats[ilat + 1],
                  clim_tropo_tps[isec + 1][ilat + 1], lat);
  return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
}
```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 1524 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;
}
```

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

Get date from day of year.

Definition at line 1542 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;
  }
}
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1570 of file libtrac.c.

```
             {

  double 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);
}
```

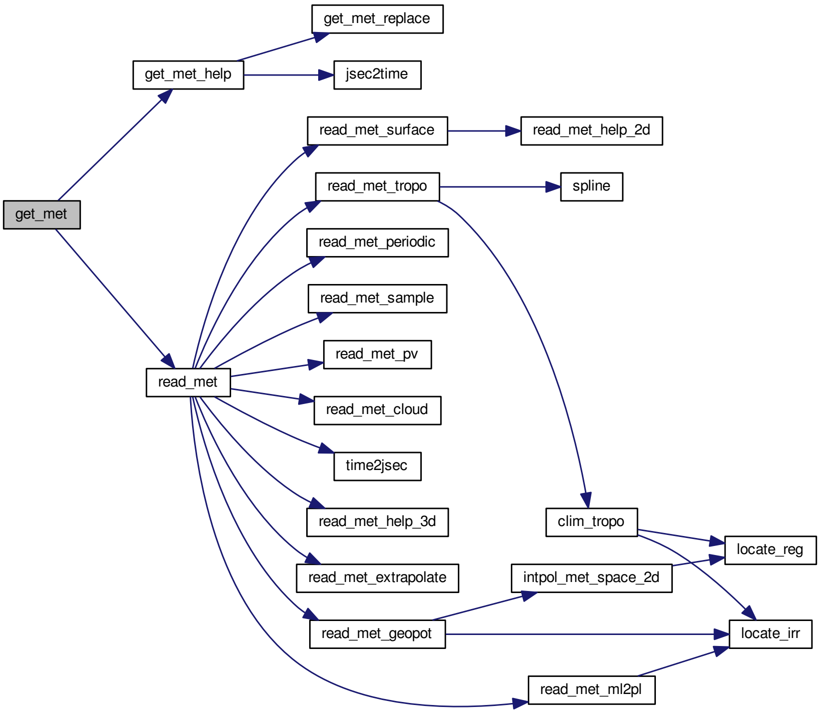
5.19.2.8 void get_met (ctl_t ∗ *ctl*, char ∗ *metbase*, double *t*, met_t ∗∗ *met0*, met_t ∗∗ *met1*)

Get meteorological data for given timestep.

Definition at line 1584 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!");
#ifdef _OPENACC
    met_t *met0up = *met0;
    met_t *met1up = *met1;
#pragma acc update device(met0up[:1],met1up[:1])
#endif
  }

  /* 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!");
#ifdef _OPENACC
    met_t *met1up = *met1;
#pragma acc update device(met1up[:1])
#endif
  }

  /* 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!");
#ifdef _OPENACC
    met_t *met0up = *met0;
#pragma acc update device(met0up[:1])
#endif
  }

  /* 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:



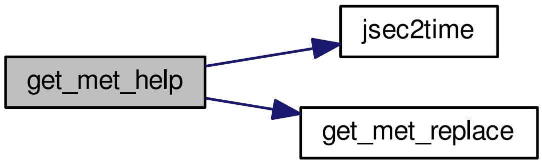
5.19.2.9 void get_met_help (double *t,* int *direct,* char ∗ *metbase,* double *dt_met,* char ∗ *filename*)

Get meteorological data for timestep.

Definition at line 1660 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:



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

Replace template strings in filename.

Definition at line 1696 of file libtrac.c.

```
                {

  char buffer[LEN], *ch;

  /* Iterate... */
  for (int 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);
  }
}
```

5.19.2.11 void intpol_met_space_3d (met_t * *met*, float *array[EX][EY][EP]*, double *p*, double *lon*, double *lat*, double * *var*, int * *ci*, double * *cw*, int *init*)

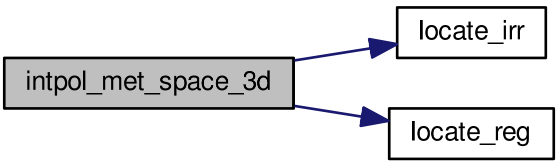
Spatial interpolation of meteorological data.

Definition at line 1719 of file libtrac.c.

```
              {

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

  /* Get interpolation indices and weights... */
  if (init) {
    ci[0] = locate_irr(met->p, met->np, p);
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);
    cw[0] = (met->p[ci[0] + 1] - p)
      / (met->p[ci[0] + 1] - met->p[ci[0]]);
    cw[1] = (met->lon[ci[1] + 1] - lon)
      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
    cw[2] = (met->lat[ci[2] + 1] - lat)
      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
  }

  /* Interpolate vertically... */
```

```
  double aux00 =
    cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
    + array[ci[1]][ci[2]][ci[0] + 1];
  double aux01 =
    cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
             array[ci[1]][ci[2] + 1][ci[0] + 1])
    + array[ci[1]][ci[2] + 1][ci[0] + 1];
  double aux10 =
    cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
             array[ci[1] + 1][ci[2]][ci[0] + 1])
    + array[ci[1] + 1][ci[2]][ci[0] + 1];
  double aux11 =
    cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
    + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];

  /* Interpolate horizontally... */
  aux00 = cw[2] * (aux00 - aux01) + aux01;
  aux11 = cw[2] * (aux10 - aux11) + aux11;
  *var = cw[1] * (aux00 - aux11) + aux11;
}
```

Here is the call graph for this function:



5.19.2.12 void intpol_met_space_2d (met_t * *met*, float *array[EX][EY]*, double *lon*, double *lat*, double * *var*, int * *ci*, double * *cw*, int *init*)

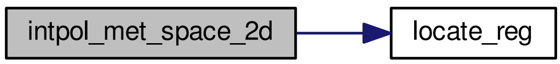
Spatial interpolation of meteorological data.

Definition at line 1773 of file libtrac.c.

```
              {

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

  /* Get interpolation indices and weights... */
  if (init) {
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);
    cw[1] = (met->lon[ci[1] + 1] - lon)
      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
    cw[2] = (met->lat[ci[2] + 1] - lat)
      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
  }

  /* Set variables... */
  double aux00 = array[ci[1]][ci[2]];
  double aux01 = array[ci[1]][ci[2] + 1];
  double aux10 = array[ci[1] + 1][ci[2]];
  double aux11 = array[ci[1] + 1][ci[2] + 1];

  /* Interpolate horizontally... */
  if (isfinite(aux00) && isfinite(aux01))
    aux00 = cw[2] * (aux00 - aux01) + aux01;
  else if (cw[2] < 0.5)
```

```
    aux00 = aux01;
  if (isfinite(aux10) && isfinite(aux11))
    aux11 = cw[2] * (aux10 - aux11) + aux11;
  else if (cw[2] > 0.5)
    aux11 = aux10;
  if (isfinite(aux00) && isfinite(aux11))
    *var = cw[1] * (aux00 - aux11) + aux11;
  else {
    if (cw[1] > 0.5)
      *var = aux00;
    else
      *var = aux11;
  }
}
```

Here is the call graph for this function:



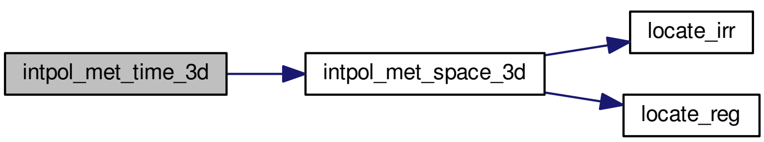
5.19.2.13 void intpol_met_time_3d (met_t * *met0*, float *array0[EX][EY][EP]*, met_t * *met1*, float *array1[EX][EY][EP]*, double *ts*, double *p*, double *lon*, double *lat*, double * *var*, int * *ci*, double * *cw*, int *init*)

Temporal interpolation of meteorological data.

Definition at line 1824 of file libtrac.c.

```
            {

  double var0, var1, wt;

  /* Spatial interpolation... */
  intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
  intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);

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

  /* Interpolate... */
  *var = wt * (var0 - var1) + var1;
}
```

Here is the call graph for this function:



5.19.2.14 void intpol_met_time_2d (met_t * *met0*, float *array0[EX][EY]*, met_t * *met1*, float *array1[EX][EY]*, double *ts*, double *lon*, double *lat*, double * *var*, int * *ci*, double * *cw*, int *init*)

Temporal interpolation of meteorological data.

Definition at line 1853 of file libtrac.c.

```
              {

  double var0, var1, wt;

  /* Spatial interpolation... */
  intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
  intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);

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

  /* Interpolate... */
  *var = wt * (var0 - var1) + var1;
}
```

Here is the call graph for this function:

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

Convert seconds to date.

Definition at line 1881 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;

  time_t 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);
}
```

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

Find array index for irregular grid.

Definition at line 1914 of file libtrac.c.

```
            {

  int ilo = 0;
  int ihi = n - 1;
  int 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;
}
```

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

Find array index for regular grid.

Definition at line 1944 of file libtrac.c.

```
            {

  /* Calculate index... */
  int 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;
}
```

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

Read atmospheric data.

Definition at line 1963 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"))) {
      WARN("File not found!");
      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;
}
```

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

Read control parameters.

Definition at line 2143 of file libtrac.c.

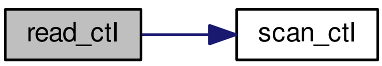
```
                {

  /* 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_lwc = -1;
  ctl->qnt_iwc = -1;
  ctl->qnt_pc = -1;
  ctl->qnt_hno3 = -1;
  ctl->qnt_oh = -1;
  ctl->qnt_rh = -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 (int 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], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
      ctl->qnt_o3 = iq;
      sprintf(ctl->qnt_unit[iq], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
      ctl->qnt_lwc = iq;
      sprintf(ctl->qnt_unit[iq], "kg/kg");
    } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
      ctl->qnt_iwc = iq;
      sprintf(ctl->qnt_unit[iq], "kg/kg");
    } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
      ctl->qnt_pc = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
      ctl->qnt_hno3 = iq;
      sprintf(ctl->qnt_unit[iq], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
      ctl->qnt_oh = iq;
      sprintf(ctl->qnt_unit[iq], "molec/cm^3");
    } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
      ctl->qnt_rh = iq;
      sprintf(ctl->qnt_unit[iq], "%%");
    } 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 (int 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, "3", NULL);
  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);

  /* Species parameters... */
  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
  if (strcmp(ctl->species, "SO2") == 0) {
    ctl->molmass = 64.066;
    ctl->oh_chem[0] = 3.3e-31;   /* (JPL Publication 15-10) */
    ctl->oh_chem[1] = 4.3;       /* (JPL Publication 15-10) */
    ctl->oh_chem[2] = 1.6e-12;   /* (JPL Publication 15-10) */
    ctl->oh_chem[3] = 0.0;       /* (JPL Publication 15-10) */
    ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
    ctl->wet_depo[1] = 0.62;    /* (FLEXPART v10.4) */
    ctl->wet_depo[2] = 1.3e-2;  /* (Sander, 2015) */
    ctl->wet_depo[3] = 2900.0;  /* (Sander, 2015) */
  } else {
    ctl->molmass =
      scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", 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);
    for (int ip = 0; ip < 4; ip++)
      ctl->oh_chem[ip] =
        scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
    for (int ip = 0; ip < 4; ip++)
      ctl->wet_depo[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO", ip, "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_stride =
    (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", 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:



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

Read meteorological data file.

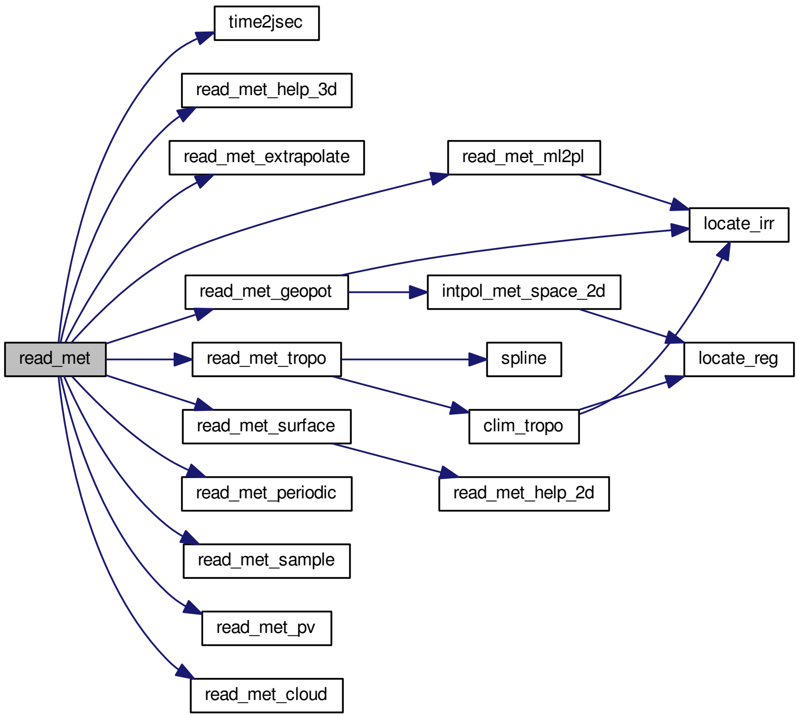
Definition at line 2455 of file libtrac.c.

```
                  {

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

```

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

  size_t np, nx, ny;

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

  /* 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);

  /* 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) {
      WARN("File not found!");
      return 0;
    }
  }

  /* 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");
    if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
      sprintf(levname, "plev");
      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... */
  if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
    ERRMSG("Cannot read temperature!");
  if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
    ERRMSG("Cannot read zonal wind!");
  if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
    ERRMSG("Cannot read meridional wind!");
  if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
    WARN("Cannot read vertical velocity");
  if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
    WARN("Cannot read specific humidity!");
  if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
    WARN("Cannot read ozone data!");
  if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
```

```
    WARN("Cannot read cloud liquid water content!");
  if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
    WARN("Cannot read cloud ice water content!");

  /* 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_3d(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);
    read_met_ml2pl(ctl, met, met->lwc);
    read_met_ml2pl(ctl, met, met->iwc);

    /* 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 data... */
  read_met_surface(ncid, met);

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

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

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

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

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

  /* Calculate cloud properties... */
  read_met_cloud(met);

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

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

Here is the call graph for this function:



5.19.2.21 void read_met_cloud (met_t * *met*)

Calculate cloud properties.

Definition at line 2622 of file libtrac.c.

```
                  {

  int ix, iy, ip;

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

      /* Init... */
      met->pc[ix][iy] = GSL_NAN;
      met->cl[ix][iy] = 0;

      /* Loop over pressure levels... */
      for (ip = 0; ip < met->np - 1; ip++) {

        /* Check pressure... */
        if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
          continue;

        /* Get cloud top pressure ... */
        if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
          met->pc[ix][iy] = (float) met->p[ip + 1];

        /* Get cloud water... */
        met->cl[ix][iy] += (float)
          (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
                  + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
           * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
      }
    }
}
```

5.19.2.22 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 2658 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 (!isfinite(met->t[ix][iy][ip0])
            || !isfinite(met->u[ix][iy][ip0])
            || !isfinite(met->v[ix][iy][ip0])
            || !isfinite(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];
        met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
        met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
      }
    }
}
```

5.19.2.23 void read_met_geopot (met_t * met)

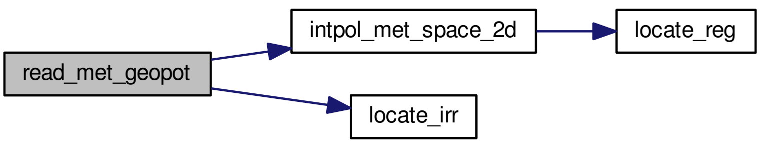
Calculate geopotential heights.

Definition at line 2692 of file libtrac.c.

```
{

  const int dx = 6, dy = 4;

  static float help[EX][EY][EP];

  double logp[EP], ts, z0, cw[3];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];

  /* Calculate log pressure... */
  for (ip = 0; ip < met->np; ip++)
    logp[ip] = log(met->p[ip]);

  /* 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;

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

      /* Get surface height... */
      intpol_met_space_2d(met, met->zs, met->lon[ix], met->
lat[iy], &z0, ci,
                          cw, 1);

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

```

      /* Get virtual temperature at the surface... */
      ts =
        LIN(met->p[ip0],
            TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
            met->p[ip0 + 1],
            TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[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 + TVIRT(met->t[ix][iy][ip0 + 1],
                                 met->h2o[ix][iy][ip0 + 1]))
                   * (log(met->ps[ix][iy]) - logp[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 *
                     (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
                      + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
                     * (logp[ip - 1] - logp[ip]));
    }

  /* Horizontal smoothing... */
#pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        n = 0;
        help[ix][iy][ip] = 0;
        for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - dy, 0);
               iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
            if (isfinite(met->z[ix3][iy2][ip])) {
              help[ix][iy][ip] += met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0)
          help[ix][iy][ip] /= (float) n;
        else
          help[ix][iy][ip] = GSL_NAN;
      }

  /* Copy 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++)
        met->z[ix][iy][ip] = help[ix][iy][ip];
}
```

Here is the call graph for this function:



5.19.2.24 int read_met_help_3d (int *ncid*, char ∗ *varname*, char ∗ *varname2*, met_t ∗ *met*, float *dest[EX][EY][EP]*, float *scl*)

Read and convert 3D variable from meteorological data file.

Definition at line 2784 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 0;

  /* Allocate... */
  ALLOC(help, float, EX * EY * EP);

  /* 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);

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

5.19.2.25 int read_met_help_2d (int *ncid*, char ∗ *varname*, char ∗ *varname2*, met_t ∗ *met*, float *dest[EX][EY]*, float *scl*)

Read and convert 2D variable from meteorological data file.

Definition at line 2828 of file libtrac.c.

```
            {

  float *help;

  int 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 0;

  /* Allocate... */
  ALLOC(help, float, EX * EY);

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

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

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

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

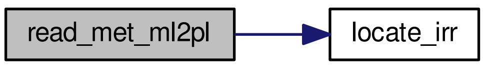
5.19.2.26 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 2871 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:



5.19.2.27 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 2910 of file libtrac.c.

```
                     {

  /* 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)
  for (int iy = 0; iy < met->ny; iy++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->zs[met->nx - 1][iy] = met->zs[0][iy];
    for (int ip = 0; ip < met->np; 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->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
      met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
      met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
    }
  }
}
```

5.19.2.28 void read_met_pv (met_t * *met*)

Calculate potential vorticity.

Definition at line 2945 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 = 0.5 * (met->lat[iy1] + met->lat[iy0]);
      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));
      }
    }
  }

  /* Fix for polar regions... */
#pragma omp parallel for default(shared) private(ix,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (ip = 0; ip < met->np; ip++) {
      met->pv[ix][0][ip]
        = met->pv[ix][1][ip]
        = met->pv[ix][2][ip];
      met->pv[ix][met->ny - 1][ip]
        = met->pv[ix][met->ny - 2][ip]
        = met->pv[ix][met->ny - 3][ip];
    }
}
```

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

Downsampling of meteorological data.

Definition at line 3045 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->zs[ix][iy] = 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->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        help->lwc[ix][iy][ip] = 0;
        help->iwc[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 - abs(ix - ix2) / ctl->met_sx)
                * (float) (1.0 - abs(iy - iy2) / ctl->met_sy)
                * (float) (1.0 - abs(ip - ip2) / ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->zs[ix][iy] += w * met->zs[ix3][iy2];
              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->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
              help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->zs[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
        help->lwc[ix][iy][ip] /= wsum;
        help->iwc[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->zs[met->nx][met->ny] = help->zs[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl->met_dp) {
        met->p[met->np] = help->p[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->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->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
        met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

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

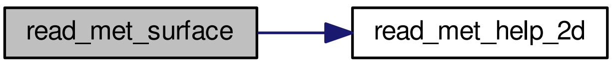
5.19.2.30 void read_met_surface (int *ncid*, met_t ∗ *met*)

Read surface data.

Definition at line 3160 of file libtrac.c.

```
                   {

  int ix, iy;

  /* Read surface pressure... */
  if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
    if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
      ERRMSG("Cannot not read surface pressure data!");
      for (ix = 0; ix < met->nx; ix++)
        for (iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) met->p[0];
    } else {
      for (iy = 0; iy < met->ny; iy++)
        for (ix = 0; ix < met->nx; ix++)
          met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
    }
  }

  /* Read geopotential height at the surface... */
  if (!read_met_help_2d
      (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
    if (!read_met_help_2d
        (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
      ERRMSG("Cannot read surface geopotential height!");
}
```

Here is the call graph for this function:



5.19.2.31 void read_met_tropo (ctl_t ∗ *ctl*, met_t ∗ *met*)

Calculate tropopause pressure.

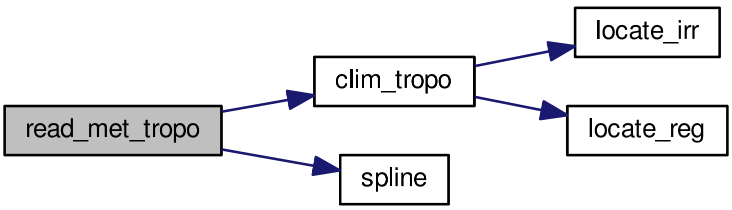
Definition at line 3190 of file libtrac.c.

```
                   {

  double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
    th2[200], z[EP], z2[200];

  int found, ix, iy, iz, iz2;

  /* Get altitude and pressure profiles... */
  for (iz = 0; iz < met->np; iz++)
    z[iz] = Z(met->p[iz]);
  for (iz = 0; iz <= 190; 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) {
#pragma omp parallel for default(shared) private(ix,iy)
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
  }

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
    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];
        spline(z, t, met->np, z2, t2, 171);

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

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
    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];
        spline(z, t, met->np, z2, t2, 191);

        /* Find 1st tropopause... */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++) {
          found = 1;
          for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
            if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
              found = 0;
              break;
            }
          if (found) {
            if (iz > 0 && iz < 170)
              met->pt[ix][iy] = (float) p2[iz];
            break;
          }
        }

        /* Find 2nd tropopause... */
        if (ctl->met_tropo == 4) {
          met->pt[ix][iy] = GSL_NAN;
          for (; iz <= 170; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 170; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
                found = 0;
                break;
              }
            if (found) {
              if (iz > 0 && iz < 170)
```

```
                met->pt[ix][iy] = (float) p2[iz];
              break;
            }
          }
        }
      }
  }

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
    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];
        spline(z, pv, met->np, z2, pv2, 171);

        /* Interpolate potential temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
        spline(z, th, met->np, z2, th2, 171);

        /* Find dynamical tropopause 3.5 PVU + 380 K */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++)
          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
            if (iz > 0 && iz < 170)
              met->pt[ix][iy] = (float) p2[iz];
            break;
          }
      }
  }

  else
    ERRMSG("Cannot calculate tropopause!");
}
```

Here is the call graph for this function:



5.19.2.32 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 3340 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);
}
```

5.19.2.33 void spline (double * *x*, double * *y*, int *n*, double * *x2*, double * *y2*, int *n2*)

Spline interpolation.

Definition at line 3412 of file libtrac.c.

```
          {

  gsl_interp_accel *acc;

  gsl_spline *s;

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);

  /* Interpolate temperature profile... */
  gsl_spline_init(s, x, y, (size_t) n);
  for (int i = 0; i < n2; i++)
    if (x2[i] <= x[0])
      y2[i] = y[0];
    else if (x2[i] >= x[n - 1])
```

```
      y2[i] = y[n - 1];
    else
      y2[i] = gsl_spline_eval(s, x2[i], acc);

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

5.19.2.34 double stddev (double * *data*, int *n*)

Calculate standard deviation.

Definition at line 3445 of file libtrac.c.

```
         {

  if (n <= 0)
    return 0;

  double avg = 0, rms = 0;

  for (int i = 0; i < n; ++i)
    avg += data[i];
  avg /= n;

  for (int i = 0; i < n; ++i)
    rms += SQR(data[i] - avg);

  return sqrt(rms / (n - 1));
}
```

5.19.2.35 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 3466 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;
}
```

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

Measure wall-clock time.

Definition at line 3497 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;
  }
}
```

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

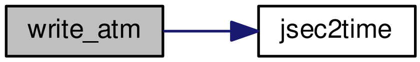
Write atmospheric data.

Definition at line 3533 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 += ctl->atm_stride) {

    /* 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:



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

Write CSI data.

Definition at line 3657 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++;
        }
      }

```

```
03805   /* Write output... */
03806   if (fmod(t, ctl->csi_dt_out) == 0) {
03807 
03808     /* Write... */
03809     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
03810             t, cx, cy, cz, cx + cy, cx + cz,
03811             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
03812             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
03813             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
03814             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
03815 
03816     /* Set counters to zero... */
03817     cx = cy = cz = 0;
03818   }
03819 
03820   /* Close file... */
03821   if (t == ctl->t_stop)
03822     fclose(out);
03823 }
```

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

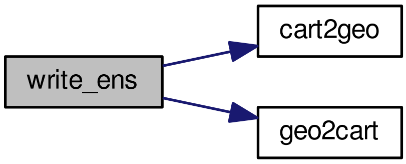
Write ensemble data.

Definition at line 3827 of file libtrac.c.

```
03831           {
03832 
03833   static FILE *out;
03834 
03835   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
03836     t0, t1, x[NENS][3], xm[3];
03837 
03838   static int ip, iq;
03839 
03840   static size_t i, n;
03841 
03842   /* Init... */
03843   if (t == ctl->t_start) {
03844 
03845     /* Check quantities... */
03846     if (ctl->qnt_ens < 0)
03847       ERRMSG("Missing ensemble IDs!");
03848 
03849     /* Create new file... */
03850     printf("Write ensemble data: %s\n", filename);
03851     if (!(out = fopen(filename, "w")))
03852       ERRMSG("Cannot create file!");
03853 
03854     /* Write header... */
03855     fprintf(out,
03856             "# $1 = time [s]\n"
03857             "# $2 = altitude [km]\n"
03858             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03859     for (iq = 0; iq < ctl->nq; iq++)
03860       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
03861               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03862     for (iq = 0; iq < ctl->nq; iq++)
03863       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
03864               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03865     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
03866   }
03867 
03868   /* Set time interval... */
03869   t0 = t - 0.5 * ctl->dt_mod;
03870   t1 = t + 0.5 * ctl->dt_mod;
03871 
03872   /* Init... */
03873   ens = GSL_NAN;
03874   n = 0;
03875 
03876   /* Loop over air parcels... */
03877   for (ip = 0; ip < atm->np; ip++) {
03878 
03879     /* Check time... */
03880     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03881       continue;
03882 
03883     /* Check ensemble id... */
03884     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:



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

Write gridded data.

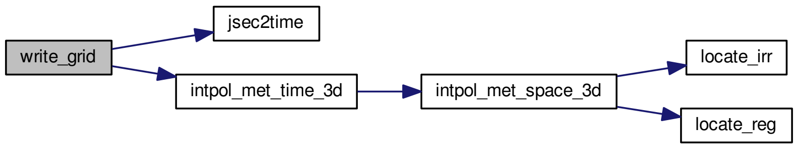
Definition at line 3958 of file libtrac.c.

```
             {

  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, cw[3];

  static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
    ci[3];

  /* 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 [ppv]\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_3d(met0, met0->t, met1, met1->t, t, press, lon,
                             lat, &temp, ci, cw, 1);

          /* 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 = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;

          /* 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:



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

Write profile data.

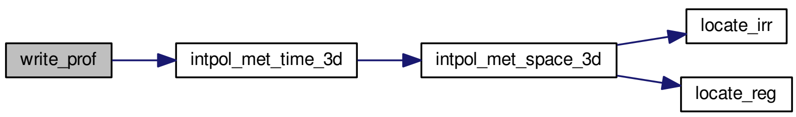
Definition at line 4112 of file libtrac.c.

```
              {

  static FILE *in, *out;

  static char line[LEN];

  static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
    t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
    o3, cw[3];

  static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];

  /* 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!");

    /* Check molar mass... */
    if (ctl->molmass <= 0)
      ERRMSG("Specify molar mass!");

    /* 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 [ppv]\n"
            "# $8 = H2O volume mixing ratio [ppv]\n"
```

```
            "# $9 = O3 volume mixing ratio [ppv]\n"
            "# $10 = observed BT index [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;
      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;
    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_3d(met0, met0->t, met1, met1->t, t, press, lon,
                             lat, &temp, ci, cw, 1);
          intpol_met_time_3d(met0, met0->h2o, met1, met1->
      h2o, t, press, lon,
                             lat, &h2o, ci, cw, 0);
          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
                             lat, &o3, ci, cw, 0);

          /* 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]);
        }
      }

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

Here is the call graph for this function:



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

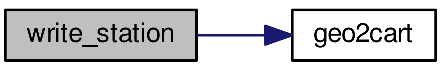
Write station data.

Definition at line 4295 of file libtrac.c.

```
            {

  static FILE *out;

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

  /* 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 (int 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 (int 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 (int 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) 2013-2020 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

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

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

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

static double clim_hno3_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
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_hno3_secs)
#endif

static double clim_hno3_lats[18] = {
  -85, -75, -65, -55, -45, -35, -25, -15, -5,
  5, 15, 25, 35, 45, 55, 65, 75, 85
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_hno3_lats)
#endif

static double clim_hno3_ps[10] = {
  4.64159, 6.81292, 10, 14.678, 21.5443,
  31.6228, 46.4159, 68.1292, 100, 146.78
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_hno3_ps)
#endif

static double clim_hno3_var[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}}
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_hno3_var)
#endif

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

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Check pressure... */
  if (p < clim_hno3_ps[0])
    p = clim_hno3_ps[0];
  else if (p > clim_hno3_ps[9])
    p = clim_hno3_ps[9];

  /* Get indices... */
  int isec = locate_irr(clim_hno3_secs, 12, sec);
  int ilat = locate_reg(clim_hno3_lats, 18, lat);
  int ip = locate_irr(clim_hno3_ps, 10, p);

  /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
  double aux00 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec][ilat][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec][ilat + 1][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec + 1][ilat][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec + 1][ilat + 1][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim_hno3_lats[ilat], aux00,
              clim_hno3_lats[ilat + 1], aux01, lat);
  aux11 = LIN(clim_hno3_lats[ilat], aux10,
              clim_hno3_lats[ilat + 1], aux11, lat);
  return LIN(clim_hno3_secs[isec], aux00,
             clim_hno3_secs[isec + 1], aux11, sec);
}

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

static double clim_oh_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
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_oh_secs)
#endif

static double clim_oh_lats[18] = {
```

```
  -85, -75, -65, -55, -45, -35, -25, -15, -5,
  5, 15, 25, 35, 45, 55, 65, 75, 85
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_oh_lats)
#endif

static double clim_oh_ps[34] = {
  0.17501, 0.233347, 0.31113, 0.41484, 0.553119, 0.737493, 0.983323,
  1.3111, 1.74813, 2.33084, 3.10779, 4.14372, 5.52496, 7.36661, 9.82214,
  13.0962, 17.4616, 23.2821, 31.0428, 41.3904, 55.1872, 73.583, 98.1107,
  130.814, 174.419, 232.559, 310.078, 413.438, 551.25, 735, 789.809,
  848.705, 911.993, 980
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_oh_ps)
#endif

static double clim_oh_var[12][18][34] = {
  {{6.422, 6.418, 7.221, 8.409, 9.768, 11.22, 12.65, 13.68, 14.03,
    13.06, 11.01, 8.791, 7.096, 6.025, 5.135, 4.057, 2.791, 1.902,
    1.318, 0.9553, 0.7083, 0.5542, 0.5145, 0.5485, 0.6292, 0.5982, 1.716,
    1.111, 0.9802, 0.6707, 0.5235, 0.4476, 0.3783, 0.3091},
   {6.311, 6.394, 7.2, 8.349, 9.664, 11.02, 12.21, 13.06, 13.28,
    12.42, 10.59, 8.552, 6.944, 5.862, 4.948, 3.826, 2.689, 1.873,
    1.302, 0.9316, 0.7053, 0.5634, 0.508, 0.5207, 0.6166, 0.6789, 1.682,
    1.218, 1.079, 0.7621, 0.6662, 0.5778, 0.4875, 0.3997},
   {5.851, 5.827, 6.393, 7.294, 8.322, 9.415, 10.46, 11.24, 11.59,
    11.13, 9.754, 7.97, 6.417, 5.331, 4.468, 3.512, 2.581, 1.855,
    1.336, 0.9811, 0.756, 0.6328, 0.6011, 0.6202, 0.7603, 0.8883, 1.303,
    1.124, 1.118, 0.9428, 0.8655, 0.8156, 0.7602, 0.6805},
   {5.276, 5.158, 5.66, 6.463, 7.419, 8.488, 9.563, 10.45, 10.94,
    10.65, 9.465, 7.762, 6.204, 5.074, 4.209, 3.324, 2.511, 1.865,
    1.386, 1.066, 0.8521, 0.723, 0.6997, 0.7492, 0.8705, 0.8088, 1.22,
    1.192, 1.298, 1.096, 1.037, 0.9589, 0.8856, 0.7726},
   {5.06, 4.919, 5.379, 6.142, 7.095, 8.156, 9.18, 10.09, 10.62,
    10.33, 9.123, 7.479, 5.967, 4.858, 3.987, 3.097, 2.342, 1.743,
    1.323, 1.044, 0.8598, 0.7596, 0.7701, 0.7858, 0.8741, 1.256, 1.266,
    1.418, 1.594, 1.247, 1.169, 1.111, 1.054, 0.9141},
   {4.921, 4.759, 5.188, 5.936, 6.847, 7.871, 8.903, 9.805, 10.31,
    10, 8.818, 7.223, 5.757, 4.66, 3.75, 2.831, 2.1, 1.579,
    1.243, 1.017, 0.8801, 0.8193, 0.9409, 1.131, 0.7313, 1.201, 1.383,
    1.643, 1.751, 1.494, 1.499, 1.647, 1.934, 2.147},
   {4.665, 4.507, 4.947, 5.652, 6.549, 7.573, 8.609, 9.499, 9.985,
    9.664, 8.478, 6.944, 5.519, 4.407, 3.511, 2.595, 1.917, 1.46,
    1.172, 1.009, 0.9372, 0.9439, 1.047, 1.219, 0.5712, 1.032, 1.342,
    1.716, 1.846, 1.551, 1.55, 1.686, 2.006, 2.235},
   {4.424, 4.288, 4.678, 5.38, 6.271, 7.291, 8.324, 9.231, 9.678,
    9.264, 8.037, 6.532, 5.141, 4.037, 3.148, 2.319, 1.715, 1.318,
    1.078, 0.9647, 0.9327, 0.9604, 1.023, 0.4157, 0.4762, 1.04, 1.589,
    2.093, 1.957, 1.557, 1.52, 1.565, 1.776, 1.904},
   {4.154, 3.996, 4.347, 5.004, 5.854, 6.869, 7.929, 8.837, 9.23,
    8.708, 7.447, 6.024, 4.761, 3.742, 2.898, 2.096, 1.55, 1.191,
    0.9749, 0.8889, 0.8745, 0.9004, 0.9648, 0.36, 0.4423, 0.973, 1.571,
    2.086, 1.971, 1.569, 1.537, 1.567, 1.74, 1.811},
   {3.862, 3.738, 4.093, 4.693, 5.499, 6.481, 7.489, 8.328, 8.637,
    8.07, 6.863, 5.56, 4.438, 3.522, 2.736, 1.971, 1.441, 1.098,
    0.8945, 0.8155, 0.7965, 0.8013, 0.8582, 1.119, 0.4076, 0.8805, 1.446,
    1.977, 1.96, 1.713, 1.793, 2.055, 2.521, 2.776},
   {3.619, 3.567, 3.943, 4.54, 5.295, 6.168, 7.033, 7.691, 7.884,
    7.326, 6.207, 5.032, 4.055, 3.263, 2.552, 1.871, 1.365, 1.022,
    0.8208, 0.7184, 0.6701, 0.6551, 0.6965, 0.7928, 0.3639, 0.6365, 0.9295,
    1.381, 1.847, 1.658, 1.668, 1.87, 2.245, 2.409},
   {3.354, 3.395, 3.811, 4.39, 5.07, 5.809, 6.514, 7, 7.054,
    6.472, 5.463, 4.466, 3.649, 2.997, 2.396, 1.785, 1.289, 0.9304,
    0.7095, 0.5806, 0.5049, 0.4639, 0.4899, 0.5149, 0.5445, 0.5185, 0.7495,
    0.8662, 1.25, 1.372, 1.384, 1.479, 1.76, 1.874},
   {3.008, 3.102, 3.503, 4.049, 4.657, 5.287, 5.845, 6.14, 6.032,
    5.401, 4.494, 3.665, 3.043, 2.575, 2.103, 1.545, 1.074, 0.7429,
    0.5514, 0.4313, 0.3505, 0.2957, 0.2688, 0.2455, 0.232, 0.3565, 0.4017,
    0.5063, 0.6618, 0.7621, 0.7915, 0.8372, 0.923, 0.9218},
   {2.548, 2.725, 3.135, 3.637, 4.165, 4.666, 5.013, 5.056, 4.72,
    4.033, 3.255, 2.64, 2.24, 1.942, 1.555, 1.085, 0.7271, 0.502,
    0.3748, 0.2897, 0.2303, 0.19, 0.1645, 0.1431, 0.1215, 0.09467, 0.1442,
    0.1847, 0.2368, 0.2463, 0.2387, 0.2459, 0.2706, 0.2751},
   {1.946, 2.135, 2.46, 2.831, 3.203, 3.504, 3.584, 3.37, 2.921,
    2.357, 1.865, 1.551, 1.392, 1.165, 0.8443, 0.5497, 0.3686, 0.2632,
    0.1978, 0.1509, 0.1197, 0.0992, 0.08402, 0.07068, 0.05652, 0.03962,
    0.03904,
    0.04357, 0.05302, 0.04795, 0.04441, 0.04296, 0.04446, 0.04576},
   {1.157, 1.285, 1.432, 1.546, 1.602, 1.556, 1.378, 1.15, 0.9351,
    0.7636, 0.6384, 0.5267, 0.4008, 0.2821, 0.2, 0.1336, 0.09109, 0.06557,
    0.05219, 0.04197, 0.03443, 0.03119, 0.02893, 0.02577, 0.02119, 0.01102,
    0.008897,
    0.00467, 0.004651, 0.004809, 0.004539, 0.004181, 0.003737, 0.002833},
```

```
    {0.07716, 0.06347, 0.05343, 0.04653, 0.0393, 0.03205, 0.02438, 0.01692,
     0.0115,
     0.007576, 0.00488, 0.002961, 0.001599, 0.001033, 0.001067, 0.001091,
     0.0005156, 0.0003818,
     0.0005061, 0.0005322, 0.0008027, 0.0008598, 0.0009114, 0.001112, 0.002042,
     0.0002528, 0.0005562,
     7.905e-06, 1.461e-07, 1.448e-07, 9.962e-08, 4.304e-08, 9.129e-17,
     1.36e-16},
    {2.613e-05, 3.434e-05, 3.646e-05, 5.101e-05, 8.027e-05, 0.0001172,
     9.886e-05, 1.933e-05, 3.14e-05,
     7.708e-05, 0.000136, 0.0001447, 0.0001049, 4.451e-05, 9.37e-06, 2.235e-06,
     1.034e-06, 4.87e-06,
     1.615e-05, 2.018e-05, 6.578e-05, 0.000178, 0.0002489, 0.0004818, 0.001231,
     0.0001402, 0.0004263,
     4.581e-07, 1.045e-12, 1.295e-13, 9.008e-14, 8.464e-14, 1.183e-13,
     2.471e-13}},
   {{5.459, 5.793, 6.743, 7.964, 9.289, 10.5, 11.39, 11.68, 11.14,
     9.663, 7.886, 6.505, 5.549, 4.931, 4.174, 3.014, 1.99, 1.339,
     0.9012, 0.6096, 0.4231, 0.3152, 0.2701, 0.2561, 0.2696, 0.2523, 0.7171,
     0.5333, 0.4876, 0.3218, 0.2536, 0.2178, 0.1861, 0.1546},
    {5.229, 5.456, 6.192, 7.112, 8.094, 9.038, 9.776, 10.16, 9.992,
     9, 7.493, 6.162, 5.154, 4.461, 3.788, 2.935, 2.058, 1.407,
     0.9609, 0.6738, 0.4989, 0.3927, 0.3494, 0.3375, 0.3689, 0.3866, 0.6716,
     0.7088, 0.6307, 0.4388, 0.3831, 0.3318, 0.2801, 0.2317},
    {4.712, 4.75, 5.283, 6.062, 6.943, 7.874, 8.715, 9.344, 9.516,
     8.913, 7.63, 6.223, 5.1, 4.346, 3.709, 2.982, 2.235, 1.605,
     1.142, 0.8411, 0.6565, 0.5427, 0.4942, 0.4907, 0.5447, 0.6331, 0.9356,
     0.7821, 0.7611, 0.663, 0.628, 0.5915, 0.5763, 0.5451},
    {4.621, 4.6, 5.091, 5.827, 6.688, 7.624, 8.529, 9.276, 9.631,
     9.219, 7.986, 6.499, 5.264, 4.401, 3.737, 2.996, 2.292, 1.69,
     1.237, 0.9325, 0.7325, 0.6093, 0.5742, 0.5871, 0.6446, 0.6139, 0.9845,
     0.9741, 1.044, 0.9091, 0.8661, 0.8218, 0.7617, 0.6884},
    {4.647, 4.573, 5.038, 5.766, 6.61, 7.534, 8.489, 9.252, 9.6,
     9.161, 7.958, 6.512, 5.259, 4.317, 3.547, 2.789, 2.13, 1.601,
     1.205, 0.9321, 0.7532, 0.6464, 0.6173, 0.5896, 0.5782, 1.014, 1.096,
     1.226, 1.387, 1.111, 1.042, 0.9908, 0.9408, 0.8311},
    {4.621, 4.534, 4.984, 5.693, 6.545, 7.49, 8.444, 9.177, 9.531,
     9.117, 7.928, 6.533, 5.27, 4.271, 3.431, 2.575, 1.902, 1.42,
     1.11, 0.9004, 0.7658, 0.6955, 0.7676, 0.9088, 0.8989, 1.028, 1.221,
     1.455, 1.583, 1.375, 1.376, 1.498, 1.744, 1.925},
    {4.514, 4.41, 4.837, 5.545, 6.416, 7.38, 8.287, 9.05, 9.416,
     9.022, 7.903, 6.496, 5.175, 4.111, 3.232, 2.38, 1.76, 1.335,
     1.068, 0.9227, 0.8515, 0.8511, 0.9534, 1.091, 0.4909, 0.9377, 1.241,
     1.592, 1.739, 1.478, 1.473, 1.597, 1.893, 2.117},
    {4.407, 4.264, 4.61, 5.263, 6.095, 7.046, 8.005, 8.805, 9.201,
     8.823, 7.705, 6.299, 4.964, 3.891, 3.046, 2.214, 1.641, 1.261,
     1.027, 0.922, 0.8759, 0.8893, 0.9782, 0.3707, 0.4349, 0.976, 1.523,
     2.021, 1.906, 1.524, 1.486, 1.53, 1.741, 1.869},
    {4.156, 4.007, 4.37, 4.987, 5.777, 6.719, 7.728, 8.578, 8.97,
     8.552, 7.409, 6.027, 4.731, 3.684, 2.814, 2.029, 1.501, 1.159,
     0.9542, 0.8666, 0.8191, 0.8371, 0.9704, 0.3324, 0.5634, 0.9279, 1.512,
     2.042, 1.951, 1.566, 1.535, 1.567, 1.739, 1.822},
    {3.98, 3.883, 4.232, 4.841, 5.594, 6.502, 7.497, 8.296, 8.631,
     8.161, 7.043, 5.703, 4.51, 3.548, 2.717, 1.951, 1.435, 1.107,
     0.9188, 0.8286, 0.772, 0.7564, 0.8263, 0.3026, 0.3854, 0.8743, 1.452,
     2.024, 2.012, 1.764, 1.85, 2.125, 2.621, 2.9},
    {3.877, 3.811, 4.167, 4.733, 5.462, 6.324, 7.144, 7.862, 8.14,
     7.695, 6.613, 5.393, 4.299, 3.42, 2.653, 1.947, 1.44, 1.1,
     0.8953, 0.7907, 0.7218, 0.6881, 0.7281, 0.8846, 0.3802, 0.6853, 1.025,
     1.593, 2.088, 1.846, 1.866, 2.108, 2.548, 2.737},
    {3.636, 3.634, 4.034, 4.637, 5.354, 6.148, 6.899, 7.495, 7.657,
     7.138, 6.107, 4.974, 3.983, 3.21, 2.558, 1.932, 1.439, 1.062,
     0.8104, 0.6632, 0.5721, 0.517, 0.5182, 0.501, 0.3937, 0.5565, 0.7675,
     0.9923, 1.455, 1.554, 1.567, 1.701, 2.077, 2.257},
    {3.366, 3.446, 3.885, 4.473, 5.16, 5.9, 6.557, 6.973, 6.973,
     6.379, 5.387, 4.385, 3.561, 2.941, 2.428, 1.848, 1.349, 0.9512,
     0.6956, 0.5432, 0.4437, 0.3733, 0.3297, 0.2972, 0.2518, 0.4613, 0.5078,
     0.6579, 0.8696, 0.9827, 1.028, 1.114, 1.275, 1.315},
    {2.975, 3.121, 3.567, 4.146, 4.797, 5.466, 6.003, 6.279, 6.087,
     5.372, 4.442, 3.614, 2.986, 2.521, 2.082, 1.532, 1.066, 0.7304,
     0.5316, 0.4123, 0.3314, 0.274, 0.2365, 0.2061, 0.1755, 0.1348, 0.2217,
     0.2854, 0.387, 0.4315, 0.4301, 0.4585, 0.5195, 0.5349},
    {2.434, 2.632, 3.054, 3.577, 4.156, 4.725, 5.1, 5.109, 4.7,
     3.958, 3.191, 2.588, 2.182, 1.866, 1.491, 1.034, 0.6936, 0.4774,
     0.3551, 0.2749, 0.2203, 0.1829, 0.1544, 0.1325, 0.1103, 0.07716, 0.08626,
     0.1037, 0.1455, 0.1418, 0.1351, 0.1339, 0.1407, 0.143},
    {1.798, 2.004, 2.333, 2.746, 3.164, 3.49, 3.572, 3.315, 2.833,
     2.269, 1.797, 1.485, 1.222, 0.9595, 0.6972, 0.4717, 0.3313, 0.2315,
     0.1727, 0.1407, 0.1204, 0.1116, 0.08791, 0.07567, 0.07432, 0.04138,
     0.02942,
     0.02407, 0.02969, 0.02808, 0.0261, 0.02364, 0.02079, 0.01623},
    {1.01, 1.1, 1.194, 1.284, 1.32, 1.278, 1.136, 0.9619, 0.815,
     0.7226, 0.6136, 0.472, 0.3124, 0.2056, 0.1448, 0.1041, 0.07233, 0.05587,
     0.05634, 0.04788, 0.03971, 0.02542, 0.01735, 0.01522, 0.01475, 0.01076,
     0.005235,
     0.003546, 0.003113, 0.003146, 0.002902, 0.002635, 0.001966, 0.001156},
    {0.04181, 0.03949, 0.03577, 0.03191, 0.02839, 0.02391, 0.01796, 0.01251,
```

```
    0.008422,
    0.005359, 0.003257, 0.001769, 0.001092, 0.0008405, 0.001744, 0.0003061,
    0.0002956, 0.000334,
    0.0004223, 0.0004062, 0.0003886, 0.0003418, 0.0003588, 0.0005709,
    0.001368, 0.0001472, 0.0003282,
    2.039e-06, 9.891e-13, 1.597e-13, 9.622e-14, 1.167e-13, 1.706e-13,
    3.59e-13}},
  {{3.361, 3.811, 4.35, 4.813, 5.132, 5.252, 5.019, 4.545, 3.913,
    3.25, 2.739, 2.417, 2.071, 1.582, 1.059, 0.6248, 0.3911, 0.2807,
    0.2095, 0.1539, 0.118, 0.0953, 0.08084, 0.07456, 0.07567, 0.07108, 0.1196,
    0.08475, 0.0839, 0.05724, 0.04673, 0.0412, 0.03626, 0.03134},
   {3.193, 3.459, 3.973, 4.578, 5.172, 5.677, 5.945, 5.913, 5.487,
    4.686, 3.838, 3.217, 2.773, 2.409, 1.9, 1.33, 0.899, 0.64,
    0.4734, 0.355, 0.277, 0.2245, 0.1928, 0.1794, 0.1806, 0.1934, 0.3268,
    0.2217, 0.1969, 0.1375, 0.1206, 0.103, 0.08729, 0.07332},
   {3.456, 3.617, 4.127, 4.778, 5.448, 6.083, 6.59, 6.858, 6.716,
    6.004, 4.975, 4.092, 3.442, 2.986, 2.474, 1.87, 1.328, 0.9231,
    0.6642, 0.5044, 0.4019, 0.336, 0.3049, 0.2996, 0.3227, 0.3571, 0.4295,
    0.3789, 0.3663, 0.3286, 0.3128, 0.3015, 0.302, 0.2968},
   {3.743, 3.817, 4.311, 4.948, 5.666, 6.417, 7.077, 7.528, 7.551,
    6.893, 5.766, 4.721, 3.913, 3.326, 2.788, 2.144, 1.555, 1.093,
    0.7905, 0.6101, 0.4943, 0.4214, 0.3965, 0.3958, 0.4205, 0.4093, 0.5808,
    0.5876, 0.6359, 0.5956, 0.5731, 0.5485, 0.512, 0.4733},
   {4.012, 4.012, 4.455, 5.113, 5.866, 6.661, 7.41, 7.98, 8.111,
    7.509, 6.364, 5.223, 4.308, 3.602, 2.951, 2.23, 1.625, 1.168,
    0.8748, 0.6916, 0.5758, 0.5122, 0.5053, 0.444, 0.4277, 0.7575, 0.7756,
    0.8848, 1.034, 0.8886, 0.8352, 0.7965, 0.7562, 0.68},
   {4.136, 4.113, 4.549, 5.187, 5.93, 6.754, 7.592, 8.298, 8.546,
    8.022, 6.882, 5.668, 4.625, 3.778, 3.016, 2.236, 1.625, 1.193,
    0.9253, 0.756, 0.6523, 0.598, 0.6186, 0.6287, 0.4804, 0.8007, 0.9732,
    1.175, 1.329, 1.194, 1.193, 1.288, 1.47, 1.603},
   {4.185, 4.119, 4.528, 5.186, 5.95, 6.801, 7.678, 8.404, 8.709,
    8.272, 7.179, 5.922, 4.767, 3.814, 2.996, 2.187, 1.599, 1.203,
    0.9596, 0.8307, 0.7617, 0.7489, 0.822, 0.9684, 0.4374, 0.7664, 1.044,
    1.36, 1.525, 1.331, 1.334, 1.447, 1.685, 1.862},
   {4.213, 4.111, 4.507, 5.117, 5.882, 6.773, 7.67, 8.464, 8.829,
    8.438, 7.378, 6.088, 4.84, 3.802, 2.915, 2.114, 1.561, 1.195,
    0.9731, 0.8773, 0.8296, 0.8295, 0.9193, 1.114, 0.3986, 0.8793, 1.394,
    1.884, 1.786, 1.437, 1.403, 1.446, 1.64, 1.752},
   {4.216, 4.092, 4.458, 5.04, 5.78, 6.67, 7.617, 8.47, 8.883,
    8.531, 7.504, 6.163, 4.862, 3.792, 2.891, 2.063, 1.519, 1.174,
    0.979, 0.8935, 0.8377, 0.8354, 0.9357, 0.3235, 0.4057, 0.9346, 1.53,
    2.078, 1.951, 1.555, 1.525, 1.557, 1.737, 1.825},
   {4.168, 4.054, 4.417, 5.011, 5.76, 6.655, 7.601, 8.453, 8.882,
    8.539, 7.487, 6.143, 4.853, 3.809, 2.92, 2.088, 1.531, 1.181,
    0.9861, 0.8955, 0.8297, 0.8116, 0.8945, 1.137, 0.4003, 0.9318, 1.557,
    2.163, 2.095, 1.82, 1.931, 2.246, 2.823, 3.14},
   {4.15, 4.078, 4.457, 5.044, 5.791, 6.692, 7.602, 8.343, 8.697,
    8.324, 7.282, 5.969, 4.743, 3.764, 2.926, 2.138, 1.572, 1.197,
    0.9808, 0.871, 0.7977, 0.764, 0.7663, 0.8665, 0.3969, 0.7472, 1.125,
    1.78, 2.277, 1.98, 2.036, 2.365, 2.936, 3.186},
   {4.028, 4.006, 4.421, 5.039, 5.793, 6.657, 7.515, 8.216, 8.5,
    8.071, 7.01, 5.768, 4.638, 3.752, 3.021, 2.305, 1.714, 1.261,
    0.9646, 0.7989, 0.6935, 0.6314, 0.6426, 0.5914, 0.4424, 0.6827, 0.917,
    1.27, 1.864, 1.895, 1.922, 2.172, 2.772, 3.141},
   {3.825, 3.864, 4.324, 4.97, 5.727, 6.565, 7.363, 7.978, 8.162,
    7.628, 6.542, 5.369, 4.372, 3.601, 2.977, 2.292, 1.697, 1.211,
    0.8808, 0.6852, 0.562, 0.4717, 0.4094, 0.366, 0.3093, 0.5639, 0.649,
    0.8282, 1.144, 1.27, 1.346, 1.522, 1.831, 1.939},
   {3.493, 3.611, 4.102, 4.736, 5.491, 6.333, 7.098, 7.615, 7.666,
    7.006, 5.88, 4.791, 3.918, 3.283, 2.733, 2.099, 1.522, 1.066,
    0.7595, 0.5845, 0.4736, 0.3918, 0.337, 0.2962, 0.2564, 0.2024, 0.3657,
    0.4744, 0.6366, 0.76, 0.8059, 0.9015, 1.065, 1.126},
   {3.167, 3.353, 3.857, 4.499, 5.225, 6.002, 6.699, 7.106, 6.98,
    6.192, 5.094, 4.125, 3.4, 2.87, 2.382, 1.774, 1.246, 0.8549,
    0.6089, 0.4666, 0.3762, 0.3082, 0.2643, 0.2283, 0.1942, 0.142, 0.1945,
    0.2512, 0.3593, 0.4048, 0.4081, 0.4206, 0.4616, 0.4738},
   {2.839, 3.07, 3.565, 4.17, 4.846, 5.529, 6.041, 6.194, 5.833,
    5.009, 4.027, 3.239, 2.656, 2.227, 1.794, 1.282, 0.9003, 0.6219,
    0.4405, 0.3353, 0.2672, 0.2205, 0.1979, 0.1911, 0.1846, 0.1044, 0.09845,
    0.1069, 0.1547, 0.1611, 0.156, 0.1466, 0.1369, 0.1212},
   {2.471, 2.75, 3.188, 3.712, 4.27, 4.77, 4.987, 4.802, 4.251,
    3.473, 2.731, 2.217, 1.83, 1.498, 1.137, 0.7594, 0.5383, 0.3859,
    0.2842, 0.2242, 0.1817, 0.1654, 0.1508, 0.1422, 0.1274, 0.07194, 0.05548,
    0.04058, 0.05226, 0.05347, 0.04942, 0.04415, 0.03177, 0.01923},
   {2.192, 2.463, 2.735, 2.958, 3.106, 3.134, 2.927, 2.561, 2.149,
    1.825, 1.627, 1.429, 1.102, 0.766, 0.4943, 0.3166, 0.2184, 0.1471,
    0.1087, 0.08747, 0.07338, 0.06265, 0.05554, 0.05001, 0.04253, 0.03353,
    0.01781,
    0.01332, 0.01371, 0.01531, 0.01438, 0.01296, 0.00982, 0.006101}},
  {{0.2905, 0.282, 0.2639, 0.2366, 0.2078, 0.1879, 0.1644, 0.1407, 0.1163,
    0.08898, 0.06233, 0.04005, 0.02438, 0.01319, 0.007679, 0.005681, 0.004482,
    0.004329,
    0.004242, 0.004414, 0.004655, 0.004515, 0.003998, 0.00342, 0.003203,
    0.001672, 0.00135,
    0.001376, 0.0004656, 0.0003538, 0.000242, 0.0001647, 0.0001078,
    5.702e-05},
```

```
  {1.5, 1.685, 1.901, 2.105, 2.216, 2.203, 2.048, 1.855, 1.609,
   1.357, 1.162, 1.012, 0.8042, 0.573, 0.372, 0.223, 0.144, 0.1095,
   0.08876, 0.07152, 0.05912, 0.05031, 0.04379, 0.03983, 0.03757, 0.03481,
   0.04463,
   0.02747, 0.02318, 0.01802, 0.0155, 0.01347, 0.01136, 0.009558},
  {2.228, 2.464, 2.889, 3.347, 3.773, 4.098, 4.191, 4.074, 3.7,
   3.132, 2.591, 2.217, 1.929, 1.594, 1.177, 0.7799, 0.5288, 0.3786,
   0.2805, 0.2144, 0.1695, 0.1403, 0.1236, 0.1157, 0.1134, 0.1148, 0.1127,
   0.1059, 0.0982, 0.08934, 0.08645, 0.08666, 0.09037, 0.09339},
  {2.761, 2.942, 3.406, 3.981, 4.592, 5.177, 5.591, 5.699, 5.391,
   4.682, 3.883, 3.267, 2.804, 2.378, 1.889, 1.347, 0.9255, 0.6382,
   0.4661, 0.361, 0.2904, 0.2459, 0.2237, 0.2143, 0.2142, 0.1959, 0.2522,
   0.2552, 0.2714, 0.274, 0.2676, 0.2624, 0.2589, 0.2436},
  {3.207, 3.317, 3.769, 4.373, 5.05, 5.759, 6.356, 6.677, 6.533,
   5.82, 4.861, 4.056, 3.414, 2.865, 2.322, 1.708, 1.204, 0.8386,
   0.6195, 0.4861, 0.3997, 0.3459, 0.32, 0.2881, 0.251, 0.4026, 0.4166,
   0.4798, 0.5723, 0.5711, 0.5498, 0.5348, 0.5181, 0.4851},
  {3.509, 3.549, 3.992, 4.607, 5.301, 6.054, 6.784, 7.302, 7.372,
   6.753, 5.706, 4.725, 3.909, 3.223, 2.58, 1.894, 1.356, 0.9716,
   0.7397, 0.5944, 0.4998, 0.4401, 0.4193, 0.3649, 0.3204, 0.5317, 0.5992,
   0.7541, 0.9257, 0.9193, 0.9167, 0.976, 1.081, 1.167},
  {3.748, 3.717, 4.128, 4.736, 5.472, 6.31, 7.086, 7.725, 7.909,
   7.387, 6.323, 5.206, 4.217, 3.376, 2.638, 1.912, 1.382, 1.026,
   0.8072, 0.6849, 0.6123, 0.5755, 0.6121, 0.6279, 0.3644, 0.5863, 0.823,
   1.068, 1.241, 1.134, 1.134, 1.222, 1.397, 1.55},
  {3.966, 3.863, 4.239, 4.826, 5.583, 6.465, 7.317, 8.033, 8.34,
   7.916, 6.843, 5.63, 4.49, 3.524, 2.698, 1.933, 1.409, 1.074,
   0.8747, 0.7756, 0.7211, 0.7103, 0.7513, 0.3147, 0.3627, 0.7582, 1.206,
   1.64, 1.639, 1.321, 1.285, 1.334, 1.496, 1.611},
  {4.184, 4.066, 4.42, 4.981, 5.712, 6.597, 7.502, 8.305, 8.719,
   8.382, 7.314, 6.02, 4.76, 3.702, 2.787, 1.993, 1.461, 1.13,
   0.9379, 0.8456, 0.7865, 0.7693, 0.835, 0.3132, 0.4006, 0.9192, 1.486,
   1.956, 1.836, 1.485, 1.462, 1.502, 1.65, 1.74},
  {4.304, 4.184, 4.566, 5.178, 5.931, 6.823, 7.761, 8.572, 9.013,
   8.72, 7.64, 6.283, 4.975, 3.89, 2.959, 2.123, 1.552, 1.194,
   0.9949, 0.8938, 0.8268, 0.814, 0.8837, 0.3629, 0.449, 1.027, 1.698,
   2.339, 2.232, 1.709, 1.64, 1.636, 1.794, 1.865},
  {4.354, 4.279, 4.704, 5.365, 6.153, 7.049, 7.975, 8.793, 9.218,
   8.889, 7.805, 6.42, 5.093, 4.013, 3.112, 2.283, 1.671, 1.263,
   1.032, 0.9124, 0.8364, 0.8112, 0.8709, 0.9784, 0.5043, 0.9917, 1.587,
   2.354, 2.467, 1.85, 1.738, 1.724, 1.873, 1.918},
  {4.35, 4.331, 4.785, 5.473, 6.289, 7.199, 8.133, 8.898, 9.288,
   8.907, 7.796, 6.411, 5.112, 4.065, 3.212, 2.421, 1.788, 1.331,
   1.04, 0.8749, 0.769, 0.7242, 0.7799, 0.8189, 0.5724, 0.86, 1.151,
   1.587, 1.946, 1.722, 1.71, 1.845, 2.146, 2.307},
  {4.268, 4.264, 4.774, 5.45, 6.298, 7.255, 8.148, 8.907, 9.242,
   8.793, 7.612, 6.244, 5.029, 4.124, 3.411, 2.65, 1.98, 1.442,
   1.074, 0.8493, 0.7077, 0.6157, 0.5826, 0.5377, 0.4848, 0.7245, 0.8927,
   1.153, 1.565, 1.551, 1.638, 1.846, 2.153, 2.304},
  {4.119, 4.16, 4.652, 5.372, 6.224, 7.158, 8.06, 8.773, 9.034,
   8.507, 7.307, 5.989, 4.866, 4.046, 3.387, 2.637, 1.959, 1.402,
   1.01, 0.7722, 0.6266, 0.527, 0.462, 0.4124, 0.3452, 0.6418, 0.572,
   0.7352, 1.009, 1.245, 1.364, 1.556, 1.828, 1.914},
  {3.932, 4.041, 4.572, 5.285, 6.144, 7.099, 7.985, 8.671, 8.849,
   8.203, 6.936, 5.657, 4.621, 3.877, 3.266, 2.526, 1.845, 1.302,
   0.9214, 0.6906, 0.5486, 0.4553, 0.4002, 0.3533, 0.3033, 0.24, 0.4062,
   0.5084, 0.7133, 0.9299, 0.9714, 1.073, 1.204, 1.211},
  {3.882, 4.074, 4.658, 5.414, 6.269, 7.182, 8.05, 8.635, 8.639,
   7.828, 6.529, 5.303, 4.339, 3.626, 3.001, 2.262, 1.617, 1.128,
   0.7869, 0.574, 0.4488, 0.3713, 0.3216, 0.2794, 0.2403, 0.1962, 0.2638,
   0.3018, 0.4483, 0.5618, 0.5717, 0.5812, 0.5993, 0.5956},
  {4.195, 4.559, 5.287, 6.13, 7.024, 7.911, 8.623, 8.913, 8.545,
   7.44, 6.059, 4.92, 4.063, 3.391, 2.747, 1.953, 1.327, 0.9036,
   0.6235, 0.4566, 0.3534, 0.2935, 0.2547, 0.2226, 0.1871, 0.1494, 0.1959,
   0.1797, 0.2736, 0.3214, 0.3054, 0.2803, 0.2443, 0.1504},
  {4.644, 5.211, 6.192, 7.373, 8.534, 9.582, 10.35, 10.37, 9.49,
   7.907, 6.299, 5.088, 4.224, 3.492, 2.701, 1.795, 1.184, 0.8001,
   0.5348, 0.3789, 0.2861, 0.2435, 0.2055, 0.1754, 0.1605, 0.1076, 0.1644,
   0.09069, 0.1624, 0.2356, 0.223, 0.2001, 0.164, 0.09807}},
 {{0.0001362, 0.0001753, 0.0001792, 0.0001881, 0.0002233, 0.0002056,
   0.0001083, 1.505e-05, 1.073e-05,
   1.22e-05, 7.475e-06, 3.518e-06, 1.292e-06, 3.812e-07, 8.611e-08,
   3.148e-09, 1.29e-09, 2.111e-08,
   7.591e-07, 1.125e-06, 4.905e-06, 1.473e-05, 3.597e-05, 9.35e-05,
   0.0002623, 0.0001066, 0.0002064,
   0.0001924, 6.532e-10, 3.205e-10, 3.838e-10, 3.541e-10, 3.331e-10,
   2.774e-10},
  {0.1865, 0.1757, 0.1596, 0.1454, 0.1343, 0.125, 0.1033, 0.08324, 0.06202,
   0.04341, 0.0295, 0.0187, 0.01129, 0.007115, 0.004932, 0.003824, 0.002733,
   0.003019,
   0.004022, 0.004428, 0.004414, 0.003826, 0.003009, 0.002117, 0.001375,
   0.0009066, 0.001031,
   0.0004673, 0.0002194, 0.0001683, 0.0001154, 6.607e-05, 3.128e-05,
   1.601e-05},
  {1.238, 1.384, 1.576, 1.788, 1.95, 1.993, 1.845, 1.593, 1.329,
   1.098, 0.9387, 0.8174, 0.6531, 0.4579, 0.2899, 0.1818, 0.1253, 0.09461,
   0.0741, 0.05946, 0.04989, 0.04353, 0.03862, 0.03469, 0.03181, 0.02862,
```

```
    0.02499,
    0.02062, 0.0178, 0.01681, 0.0162, 0.01679, 0.01921, 0.02192},
   {1.92, 2.106, 2.478, 2.931, 3.421, 3.86, 4.043, 3.866, 3.41,
    2.84, 2.372, 2.077, 1.825, 1.472, 1.07, 0.7023, 0.4688, 0.3296,
    0.2453, 0.1889, 0.15, 0.1246, 0.1094, 0.09788, 0.09079, 0.08267, 0.08977,
    0.09461, 0.09867, 0.09977, 0.09675, 0.09757, 0.1011, 0.1004},
   {2.505, 2.661, 3.095, 3.659, 4.295, 4.928, 5.368, 5.413, 5.018,
    4.291, 3.582, 3.052, 2.636, 2.193, 1.714, 1.199, 0.8121, 0.5616,
    0.4193, 0.3284, 0.2656, 0.2245, 0.2009, 0.1819, 0.1615, 0.2142, 0.2142,
    0.2523, 0.3029, 0.3253, 0.3248, 0.323, 0.3186, 0.3005},
   {3.053, 3.153, 3.582, 4.16, 4.823, 5.525, 6.111, 6.38, 6.176,
    5.453, 4.575, 3.84, 3.219, 2.652, 2.09, 1.509, 1.055, 0.7478,
    0.5678, 0.4529, 0.3746, 0.322, 0.2979, 0.2602, 0.284, 0.3515, 0.39,
    0.4915, 0.6421, 0.6745, 0.684, 0.7208, 0.7815, 0.8179},
   {3.422, 3.44, 3.875, 4.477, 5.191, 5.965, 6.661, 7.123, 7.132,
    6.48, 5.463, 4.526, 3.718, 3.005, 2.344, 1.688, 1.206, 0.8837,
    0.6856, 0.5649, 0.4863, 0.445, 0.4856, 0.5006, 0.3136, 0.4515, 0.6109,
    0.7863, 0.9684, 0.937, 0.937, 0.9949, 1.106, 1.204},
   {3.782, 3.708, 4.085, 4.684, 5.42, 6.264, 7.105, 7.757, 7.965,
    7.421, 6.309, 5.169, 4.161, 3.312, 2.538, 1.814, 1.314, 0.9921,
    0.7986, 0.6906, 0.6241, 0.6037, 0.6883, 0.3135, 0.3539, 0.6781, 1.044,
    1.41, 1.465, 1.198, 1.162, 1.195, 1.315, 1.399},
   {4.093, 3.956, 4.303, 4.881, 5.64, 6.546, 7.473, 8.261, 8.617,
    8.211, 7.109, 5.806, 4.611, 3.616, 2.718, 1.923, 1.396, 1.077,
    0.8999, 0.7993, 0.7318, 0.69, 0.7736, 0.3307, 0.5444, 0.8973, 1.424,
    1.867, 1.76, 1.423, 1.397, 1.429, 1.558, 1.631},
   {4.316, 4.179, 4.572, 5.214, 6.033, 6.982, 7.93, 8.754, 9.154,
    8.809, 7.719, 6.283, 4.962, 3.898, 2.961, 2.106, 1.526, 1.178,
    0.9911, 0.8826, 0.8091, 0.792, 0.8984, 0.4081, 0.6591, 1.079, 1.762,
    2.394, 2.262, 1.722, 1.649, 1.64, 1.793, 1.856},
   {4.422, 4.322, 4.787, 5.478, 6.345, 7.323, 8.254, 9.118, 9.564,
    9.218, 8.075, 6.604, 5.203, 4.058, 3.117, 2.274, 1.677, 1.284,
    1.048, 0.9151, 0.8297, 0.7966, 0.8935, 0.5312, 0.7531, 1.072, 1.715,
    2.495, 2.546, 1.897, 1.782, 1.77, 1.93, 1.983},
   {4.547, 4.469, 4.942, 5.688, 6.549, 7.488, 8.486, 9.332, 9.775,
    9.417, 8.264, 6.785, 5.379, 4.249, 3.33, 2.505, 1.852, 1.39,
    1.098, 0.9308, 0.8263, 0.7895, 0.8576, 1.037, 0.635, 0.9803, 1.371,
    1.867, 2.133, 1.823, 1.82, 1.997, 2.372, 2.584},
   {4.687, 4.634, 5.104, 5.806, 6.687, 7.693, 8.664, 9.479, 9.89,
    9.543, 8.412, 6.918, 5.518, 4.426, 3.541, 2.711, 2.014, 1.493,
    1.144, 0.9317, 0.797, 0.7099, 0.6575, 0.6067, 0.526, 0.9154, 1.118,
    1.507, 1.832, 1.788, 1.887, 2.179, 2.647, 2.868},
   {4.695, 4.698, 5.185, 5.955, 6.871, 7.893, 8.877, 9.693, 10.1,
    9.723, 8.528, 6.981, 5.591, 4.567, 3.788, 2.967, 2.242, 1.655,
    1.222, 0.9406, 0.7599, 0.6428, 0.57, 0.513, 0.4496, 0.9249, 0.8597,
    1.064, 1.428, 1.661, 1.831, 2.151, 2.551, 2.772},
   {4.817, 4.823, 5.364, 6.154, 7.109, 8.194, 9.225, 10.09, 10.53,
    10.13, 8.82, 7.204, 5.79, 4.774, 3.99, 3.147, 2.374, 1.734,
    1.249, 0.9272, 0.7213, 0.5919, 0.522, 0.4691, 0.4205, 0.3435, 0.704,
    0.795, 1.078, 1.369, 1.506, 1.697, 1.921, 2.057},
   {5.178, 5.275, 5.904, 6.794, 7.806, 8.901, 9.941, 10.76, 11.06,
    10.41, 8.899, 7.194, 5.78, 4.767, 3.992, 3.132, 2.304, 1.622,
    1.125, 0.8041, 0.6135, 0.4964, 0.4334, 0.388, 0.3494, 0.3124, 0.5608,
    0.6202, 0.8549, 1.142, 1.185, 1.327, 1.469, 1.521},
   {5.765, 5.982, 6.854, 7.999, 9.258, 10.55, 11.75, 12.58, 12.65,
    11.51, 9.567, 7.638, 6.122, 5.016, 4.17, 3.208, 2.283, 1.528,
    0.9814, 0.6677, 0.4925, 0.3884, 0.3284, 0.2876, 0.2571, 0.2207, 0.5107,
    0.5167, 0.7005, 0.8149, 0.7878, 0.7521, 0.6899, 0.4697},
   {5.862, 6.014, 6.876, 8.048, 9.368, 10.8, 12.34, 13.65, 13.83,
    12.41, 10.19, 8.106, 6.468, 5.285, 4.422, 3.392, 2.324, 1.509,
    0.9796, 0.6834, 0.5019, 0.387, 0.3295, 0.2804, 0.2425, 0.2084, 0.4694,
    0.4497, 0.5949, 0.6504, 0.6155, 0.5696, 0.5077, 0.2744}},
  {{6.617e-05, 8.467e-05, 8.509e-05, 9.824e-05, 0.0001317, 0.0001499,
    0.0001104, 1.226e-05, 1.003e-05,
    1.345e-05, 8.036e-06, 2.28e-06, 3.166e-07, 1.803e-08, 6.079e-10,
    3.031e-10, 1.336e-09, 1.748e-08,
    3.057e-07, 8.184e-07, 1.95e-06, 2.238e-06, 4.658e-06, 1.393e-05,
    4.326e-05, 3.288e-05, 0.0001662,
    8.012e-05, 6.48e-10, 3.371e-10, 4.509e-10, 4.052e-10, 3.677e-10,
    2.996e-10},
   {0.003936, 0.002158, 0.001688, 0.001555, 0.001396, 0.0008637, 0.0003091,
    4.908e-05, 1.173e-05,
    1.326e-05, 9.097e-06, 3.324e-06, 6.336e-07, 8.553e-08, 4.851e-09,
    3.058e-09, 9.169e-09, 2.451e-07,
    8.77e-07, 2.973e-07, 2.556e-07, 2.79e-07, 8.674e-07, 4.17e-06, 2.373e-05,
    3.876e-05, 6.493e-05,
    2.13e-05, 2.574e-09, 2.676e-10, 2.356e-10, 1.947e-10, 2.376e-10,
    1.955e-10},
   {0.8642, 0.927, 0.9782, 1.001, 0.9705, 0.885, 0.7521, 0.6223, 0.5357,
    0.4614, 0.373, 0.2955, 0.2138, 0.1342, 0.07775, 0.04773, 0.0348, 0.02871,
    0.02503, 0.02329, 0.02308, 0.02227, 0.01967, 0.01687, 0.01381, 0.008063,
    0.006489,
    0.005433, 0.004804, 0.00487, 0.004735, 0.004696, 0.004587, 0.003634},
   {1.584, 1.731, 2.006, 2.368, 2.742, 3.047, 3.124, 2.894, 2.517,
    2.087, 1.761, 1.588, 1.399, 1.091, 0.7549, 0.4844, 0.3282, 0.2357,
    0.1737, 0.1299, 0.1001, 0.08162, 0.0699, 0.06056, 0.05333, 0.07014,
    0.04552,
```

```
    0.04907, 0.04781, 0.04938, 0.04864, 0.04971, 0.05141, 0.05078},
   {2.256, 2.4, 2.786, 3.287, 3.854, 4.41, 4.773, 4.776, 4.396,
    3.755, 3.16, 2.747, 2.381, 1.94, 1.458, 0.9865, 0.6576, 0.4582,
    0.3447, 0.2672, 0.2121, 0.1768, 0.1538, 0.1358, 0.1219, 0.1466, 0.1371,
    0.161, 0.1896, 0.2116, 0.2133, 0.2153, 0.2145, 0.2042},
   {2.844, 2.913, 3.296, 3.841, 4.473, 5.127, 5.672, 5.933, 5.724,
    5.005, 4.187, 3.549, 3.006, 2.486, 1.928, 1.368, 0.9363, 0.6524,
    0.4946, 0.3921, 0.3189, 0.2696, 0.2503, 0.2181, 0.2227, 0.2946, 0.297,
    0.3704, 0.4834, 0.5473, 0.5565, 0.581, 0.6189, 0.6392},
   {3.309, 3.337, 3.765, 4.35, 5.025, 5.756, 6.438, 6.908, 6.947,
    6.3, 5.252, 4.33, 3.576, 2.912, 2.258, 1.618, 1.15, 0.8271,
    0.6351, 0.5129, 0.4287, 0.3846, 0.4351, 0.4889, 0.3229, 0.4124, 0.4839,
    0.6285, 0.8468, 0.9091, 0.9059, 0.9413, 1.007, 1.074},
   {3.725, 3.662, 4.042, 4.647, 5.381, 6.204, 7.019, 7.636, 7.841,
    7.318, 6.177, 5.04, 4.09, 3.299, 2.545, 1.822, 1.303, 0.967,
    0.7683, 0.6506, 0.5816, 0.5712, 0.667, 0.3774, 0.3833, 0.5497, 0.7343,
    0.9899, 1.25, 1.213, 1.147, 1.144, 1.27, 1.389},
   {4.074, 3.916, 4.273, 4.851, 5.624, 6.564, 7.525, 8.296, 8.632,
    8.184, 7.013, 5.711, 4.593, 3.691, 2.813, 2.003, 1.445, 1.097,
    0.8982, 0.7815, 0.7103, 0.6929, 0.7841, 0.3697, 0.42, 0.8027, 1.255,
    1.716, 1.727, 1.411, 1.379, 1.416, 1.548, 1.618},
   {4.378, 4.233, 4.657, 5.297, 6.121, 7.098, 8.11, 8.937, 9.31,
    8.877, 7.696, 6.289, 5.029, 3.997, 3.066, 2.189, 1.589, 1.209,
    0.9953, 0.8642, 0.7883, 0.7739, 0.9011, 0.4115, 0.6312, 0.9904, 1.555,
    2.033, 1.915, 1.547, 1.523, 1.55, 1.675, 1.75},
   {4.574, 4.464, 4.871, 5.569, 6.471, 7.487, 8.486, 9.356, 9.788,
    9.387, 8.196, 6.717, 5.357, 4.245, 3.29, 2.387, 1.738, 1.316,
    1.064, 0.9101, 0.8224, 0.8068, 0.9022, 0.478, 0.7124, 1.093, 1.75,
    2.355, 2.233, 1.721, 1.663, 1.652, 1.836, 1.906},
   {4.771, 4.625, 5.041, 5.796, 6.698, 7.684, 8.72, 9.611, 10.08,
    9.736, 8.573, 7.013, 5.571, 4.447, 3.501, 2.606, 1.915, 1.439,
    1.139, 0.956, 0.8517, 0.8306, 0.9424, 1.177, 0.5869, 1.036, 1.44,
    1.878, 2.039, 1.705, 1.737, 1.923, 2.385, 2.641},
   {5.006, 4.866, 5.324, 6.09, 7.006, 8.036, 9.084, 9.893, 10.33,
    10.02, 8.839, 7.228, 5.736, 4.611, 3.71, 2.8, 2.063, 1.532,
    1.183, 0.9673, 0.8363, 0.7853, 0.8359, 0.8923, 0.569, 1.039, 1.365,
    1.742, 1.904, 1.758, 1.875, 2.251, 2.843, 3.224},
   {5.099, 4.961, 5.447, 6.211, 7.152, 8.22, 9.322, 10.2, 10.72,
    10.48, 9.248, 7.559, 5.993, 4.842, 3.975, 3.116, 2.35, 1.736,
    1.297, 1.003, 0.8134, 0.702, 0.6582, 0.6213, 0.5842, 0.5417, 1.158,
    1.4, 1.637, 1.863, 2.063, 2.497, 3.122, 3.523},
   {5.329, 5.217, 5.717, 6.57, 7.585, 8.724, 9.852, 10.78, 11.29,
    10.97, 9.671, 7.914, 6.3, 5.116, 4.263, 3.389, 2.58, 1.892,
    1.383, 1.036, 0.8131, 0.6799, 0.6149, 0.5698, 0.5283, 0.4671, 1.093,
    1.188, 1.482, 1.733, 1.908, 2.209, 2.612, 2.814},
   {6.071, 6.113, 6.75, 7.703, 8.826, 10.05, 11.16, 11.96, 12.25,
    11.61, 10.02, 8.102, 6.438, 5.24, 4.366, 3.448, 2.553, 1.795,
    1.245, 0.8829, 0.6737, 0.5561, 0.4996, 0.4579, 0.4257, 0.4016, 0.968,
    1.042, 1.284, 1.504, 1.639, 1.849, 2.098, 2.255},
   {6.417, 6.432, 7.23, 8.422, 9.846, 11.4, 12.82, 13.78, 14,
    12.98, 10.94, 8.756, 6.951, 5.69, 4.77, 3.716, 2.672, 1.818,
    1.224, 0.8517, 0.6496, 0.5355, 0.4766, 0.4254, 0.389, 0.351, 0.9729,
    0.9755, 1.108, 1.026, 0.9967, 0.9687, 0.9329, 0.8183},
   {6.462, 6.445, 7.221, 8.396, 9.796, 11.38, 12.96, 14.19, 14.59,
    13.52, 11.33, 9.046, 7.189, 5.92, 5.015, 3.949, 2.77, 1.841,
    1.239, 0.86, 0.6441, 0.5179, 0.446, 0.42, 0.3818, 0.3068, 0.9267,
    0.8849, 0.9828, 0.8458, 0.7544, 0.6963, 0.6233, 0.5358}},
  {{3.24e-05, 4.086e-05, 4.221e-05, 5.381e-05, 8.808e-05, 0.0001261,
    0.0001263, 2.932e-05, 2.057e-05,
    4.321e-05, 3.43e-05, 1.213e-05, 1.862e-06, 1.067e-07, 2.983e-09,
    2.578e-08, 2.254e-09, 1.504e-08,
    1.599e-07, 1.523e-07, 2.033e-07, 5.271e-07, 1.417e-06, 4.518e-06,
    1.358e-05, 9.266e-06, 0.0001569,
    3.355e-05, 7.653e-10, 3.889e-10, 5.082e-10, 4.498e-10, 4.072e-10,
    3.302e-10},
   {0.0688, 0.05469, 0.04502, 0.03604, 0.02861, 0.02128, 0.01453, 0.00924,
    0.00574,
    0.003447, 0.001888, 0.001147, 0.001154, 0.001548, 0.001193, 6.955e-05,
    0.0003258, 1.673e-05,
    1.393e-05, 1.849e-05, 2.841e-05, 4.045e-05, 5.864e-05, 4.426e-05,
    1.722e-05, 3.186e-05, 5.582e-05,
    6.823e-06, 3.559e-09, 3.282e-10, 2.847e-10, 2.3e-10, 2.728e-10,
    2.237e-10},
   {1.056, 1.158, 1.28, 1.377, 1.429, 1.395, 1.242, 1.043, 0.8709,
    0.7435, 0.6183, 0.4824, 0.3665, 0.2668, 0.1749, 0.098, 0.09722, 0.08524,
    0.05545, 0.04501, 0.04476, 0.04491, 0.04231, 0.03868, 0.03336, 0.01108,
    0.01596,
    0.009172, 0.008499, 0.008763, 0.008672, 0.008872, 0.008926, 0.007552},
   {1.798, 1.952, 2.239, 2.589, 2.958, 3.261, 3.328, 3.12, 2.741,
    2.293, 1.917, 1.686, 1.53, 1.28, 0.9362, 0.6052, 0.4093, 0.294,
    0.2166, 0.1609, 0.1206, 0.09532, 0.08094, 0.07007, 0.06252, 0.1111,
    0.04996,
    0.0572, 0.0583, 0.06137, 0.06077, 0.0616, 0.0644, 0.06454},
   {2.42, 2.566, 2.952, 3.45, 3.987, 4.499, 4.853, 4.878, 4.55,
    3.918, 3.264, 2.802, 2.467, 2.094, 1.632, 1.127, 0.7523, 0.5168,
    0.386, 0.2982, 0.2346, 0.1928, 0.1694, 0.1534, 0.1384, 0.1672, 0.1434,
    0.1726, 0.2115, 0.2414, 0.2445, 0.2487, 0.2496, 0.2381},
```

```
{3.026, 3.096, 3.501, 4.016, 4.608, 5.244, 5.786, 6.088, 5.975,
 5.326, 4.429, 3.701, 3.142, 2.636, 2.071, 1.477, 1.015, 0.7056,
 0.5315, 0.4215, 0.3413, 0.2869, 0.2681, 0.2507, 0.2616, 0.3004, 0.3042,
 0.3879, 0.5203, 0.6045, 0.6196, 0.651, 0.6986, 0.7171},
{3.463, 3.475, 3.878, 4.441, 5.112, 5.838, 6.522, 7.015, 7.096,
 6.52, 5.487, 4.496, 3.734, 3.105, 2.456, 1.776, 1.26, 0.9016,
 0.6841, 0.5485, 0.454, 0.404, 0.4609, 0.5432, 0.376, 0.4702, 0.5045,
 0.6555, 0.9157, 1.076, 1.113, 1.216, 1.372, 1.497},
{3.745, 3.679, 4.071, 4.666, 5.394, 6.22, 7.064, 7.725, 7.961,
 7.448, 6.321, 5.161, 4.233, 3.472, 2.73, 1.96, 1.409, 1.04,
 0.8213, 0.6863, 0.6068, 0.5872, 0.7122, 0.9507, 0.4302, 0.5845, 0.768,
 1.036, 1.308, 1.372, 1.393, 1.57, 2.023, 2.387},
{4.046, 3.927, 4.286, 4.869, 5.645, 6.599, 7.567, 8.4, 8.74,
 8.247, 7.036, 5.735, 4.638, 3.741, 2.905, 2.066, 1.493, 1.132,
 0.9307, 0.7993, 0.7306, 0.7334, 0.8811, 1.197, 0.4369, 0.7932, 1.239,
 1.716, 1.764, 1.639, 1.738, 2.042, 2.543, 2.814},
{4.303, 4.169, 4.566, 5.224, 6.055, 7.053, 8.124, 9.037, 9.471,
 8.998, 7.714, 6.277, 4.981, 3.935, 3.018, 2.182, 1.599, 1.224,
 1.01, 0.8634, 0.7916, 0.7955, 0.9234, 1.237, 0.6362, 0.9588, 1.513,
 1.996, 1.902, 1.545, 1.53, 1.572, 1.712, 1.768},
{4.506, 4.381, 4.819, 5.529, 6.428, 7.462, 8.518, 9.453, 9.907,
 9.467, 8.237, 6.703, 5.29, 4.149, 3.213, 2.346, 1.732, 1.324,
 1.075, 0.9181, 0.8397, 0.8375, 0.9434, 0.5018, 0.5414, 1.083, 1.74,
 2.347, 2.227, 1.721, 1.661, 1.648, 1.831, 1.912},
{4.687, 4.558, 4.978, 5.725, 6.658, 7.731, 8.789, 9.675, 10.13,
 9.748, 8.526, 6.958, 5.498, 4.351, 3.431, 2.537, 1.87, 1.415,
 1.127, 0.9521, 0.8587, 0.8423, 0.8885, 1.184, 0.5938, 1.064, 1.469,
 1.877, 2.006, 1.669, 1.7, 1.879, 2.347, 2.626},
{4.936, 4.809, 5.237, 6.008, 6.937, 7.97, 9.016, 9.924, 10.41,
 10.04, 8.758, 7.153, 5.667, 4.529, 3.633, 2.737, 2.011, 1.493,
 1.161, 0.9601, 0.8439, 0.8078, 0.8976, 1.023, 0.6551, 1.239, 1.595,
 1.9, 1.923, 1.721, 1.833, 2.227, 2.857, 3.283},
{5.075, 4.962, 5.428, 6.241, 7.2, 8.253, 9.328, 10.2, 10.65,
 10.28, 8.983, 7.308, 5.772, 4.63, 3.752, 2.906, 2.151, 1.57,
 1.188, 0.9525, 0.8028, 0.7168, 0.7021, 0.7201, 0.7339, 0.6036, 1.423,
 1.574, 1.715, 1.841, 2.05, 2.525, 3.214, 3.663},
{5.33, 5.234, 5.731, 6.529, 7.518, 8.638, 9.699, 10.59, 11.06,
 10.69, 9.363, 7.602, 6.028, 4.875, 4.029, 3.2, 2.426, 1.77,
 1.305, 0.9937, 0.7977, 0.6831, 0.6247, 0.6058, 0.5859, 0.5171, 1.306,
 1.309, 1.552, 1.652, 1.83, 2.179, 2.619, 2.844},
{5.866, 5.848, 6.455, 7.346, 8.396, 9.533, 10.53, 11.34, 11.68,
 11.13, 9.619, 7.806, 6.227, 5.073, 4.235, 3.4, 2.554, 1.805,
 1.264, 0.9221, 0.7291, 0.623, 0.5765, 0.5499, 0.5313, 0.5058, 1.13,
 1.193, 1.384, 1.521, 1.658, 1.903, 2.204, 2.394},
{6.303, 6.398, 7.222, 8.351, 9.705, 11.15, 12.37, 13.21, 13.37,
 12.4, 10.47, 8.393, 6.679, 5.472, 4.597, 3.653, 2.67, 1.827,
 1.234, 0.8789, 0.697, 0.5977, 0.5427, 0.5065, 0.4765, 0.4416, 1.108,
 1.122, 1.2, 1.026, 0.9995, 0.9762, 0.9406, 0.8155},
{6.374, 6.361, 7.22, 8.387, 9.771, 11.27, 12.69, 13.72, 14.07,
 13.07, 10.88, 8.681, 6.947, 5.79, 4.913, 3.885, 2.773, 1.855,
 1.244, 0.9142, 0.7256, 0.598, 0.5151, 0.5035, 0.4732, 0.3585, 1.059,
 1.045, 1.089, 0.8483, 0.7697, 0.7077, 0.6288, 0.5436}},
{{0.03789, 0.03408, 0.03134, 0.02846, 0.02395, 0.01876, 0.01296, 0.008683,
 0.005595,
 0.003327, 0.001899, 0.001016, 0.0006645, 0.001147, 0.000686, 0.0006987,
 0.0006744, 0.0003903,
 1.823e-05, 1.418e-05, 1.097e-05, 9.197e-06, 8.385e-06, 5.361e-06,
 1.047e-05, 8.724e-06, 0.0001224,
 1.608e-05, 7.761e-10, 4.005e-10, 5.216e-10, 4.653e-10, 4.282e-10,
 3.472e-10},
{0.9685, 1.035, 1.103, 1.161, 1.165, 1.106, 0.9814, 0.8435, 0.7222,
 0.6291, 0.5477, 0.4518, 0.3058, 0.2105, 0.1492, 0.08369, 0.1841, 0.1379,
 0.1032, 0.06085, 0.03194, 0.02288, 0.0206, 0.02385, 0.0298, 0.0102,
 0.002722,
 0.00743, 0.007294, 0.006871, 0.006275, 0.005435, 0.004153, 0.002587},
{1.842, 2.002, 2.295, 2.618, 2.95, 3.205, 3.226, 2.966, 2.55,
 2.078, 1.668, 1.355, 1.079, 0.8142, 0.7131, 0.5158, 0.3251, 0.2412,
 0.1924, 0.1651, 0.1599, 0.164, 0.1325, 0.1215, 0.09591, 0.1117, 0.03993,
 0.04814, 0.04778, 0.04729, 0.04627, 0.0467, 0.04597, 0.04172},
{2.489, 2.643, 3.032, 3.507, 4.035, 4.533, 4.829, 4.772, 4.375,
 3.714, 3.06, 2.551, 2.207, 1.938, 1.628, 1.185, 0.8109, 0.5625,
 0.4101, 0.3078, 0.2325, 0.1813, 0.1507, 0.1311, 0.1178, 0.1964, 0.1063,
 0.1261, 0.1425, 0.1561, 0.1546, 0.1532, 0.1543, 0.1498},
{3.03, 3.156, 3.579, 4.144, 4.775, 5.42, 5.935, 6.129, 5.878,
 5.138, 4.235, 3.481, 2.956, 2.542, 2.086, 1.502, 1.028, 0.7057,
 0.5223, 0.4097, 0.3291, 0.2729, 0.2417, 0.2235, 0.2035, 0.2565, 0.2228,
 0.274, 0.339, 0.387, 0.3918, 0.396, 0.3931, 0.3692},
{3.447, 3.508, 3.934, 4.533, 5.226, 5.96, 6.611, 7.003, 6.968,
 6.331, 5.301, 4.324, 3.576, 3.009, 2.45, 1.808, 1.285, 0.894,
 0.6612, 0.5244, 0.4273, 0.3588, 0.3341, 0.3096, 0.296, 0.4036, 0.403,
 0.5114, 0.6763, 0.7735, 0.7969, 0.8461, 0.9229, 0.9621},
{3.722, 3.706, 4.103, 4.704, 5.429, 6.225, 6.997, 7.531, 7.688,
 7.197, 6.151, 5.021, 4.119, 3.436, 2.786, 2.084, 1.513, 1.082,
 0.808, 0.6428, 0.5308, 0.473, 0.5475, 0.6436, 0.4784, 0.5735, 0.5953,
 0.7959, 1.106, 1.287, 1.339, 1.478, 1.689, 1.86},
{3.921, 3.841, 4.189, 4.783, 5.529, 6.4, 7.263, 7.978, 8.293,
 7.875, 6.788, 5.549, 4.498, 3.668, 2.902, 2.117, 1.551, 1.158,
```

```
  0.9124, 0.7546, 0.6628, 0.6526, 0.7867, 0.9666, 0.4912, 0.664, 0.8688,
  1.172, 1.475, 1.537, 1.57, 1.791, 2.33, 2.77},
 {4.115, 3.995, 4.345, 4.915, 5.667, 6.583, 7.547, 8.4, 8.806,
  8.416, 7.274, 5.963, 4.777, 3.806, 2.936, 2.114, 1.555, 1.194,
  0.9788, 0.8251, 0.7444, 0.7323, 0.8067, 0.4073, 0.4371, 0.8223, 1.294,
  1.799, 1.845, 1.715, 1.83, 2.164, 2.718, 3.019},
 {4.267, 4.151, 4.545, 5.138, 5.928, 6.9, 7.906, 8.818, 9.276,
  8.888, 7.7, 6.275, 4.923, 3.83, 2.949, 2.126, 1.57, 1.211,
  0.9919, 0.8349, 0.7611, 0.7613, 0.8757, 0.4158, 0.6205, 0.9285, 1.474,
  1.966, 1.886, 1.546, 1.534, 1.58, 1.727, 1.781},
 {4.357, 4.249, 4.661, 5.37, 6.245, 7.245, 8.255, 9.142, 9.591,
  9.187, 7.974, 6.489, 5.106, 3.983, 3.055, 2.223, 1.646, 1.263,
  1.021, 0.8595, 0.7835, 0.7906, 0.9308, 0.4752, 0.6857, 1.015, 1.644,
  2.238, 2.142, 1.682, 1.628, 1.619, 1.797, 1.869},
 {4.506, 4.431, 4.893, 5.628, 6.535, 7.541, 8.516, 9.354, 9.751,
  9.305, 8.052, 6.578, 5.209, 4.113, 3.227, 2.379, 1.745, 1.32,
  1.053, 0.8816, 0.796, 0.7899, 0.9061, 1.191, 0.5672, 0.9669, 1.349,
  1.746, 1.888, 1.592, 1.62, 1.791, 2.222, 2.485},
 {4.576, 4.506, 4.983, 5.745, 6.664, 7.67, 8.657, 9.446, 9.792,
  9.306, 8.032, 6.569, 5.251, 4.219, 3.374, 2.515, 1.835, 1.359,
  1.057, 0.8733, 0.7654, 0.7338, 0.8346, 1.067, 0.6038, 1.054, 1.38,
  1.684, 1.756, 1.589, 1.679, 2.018, 2.626, 2.986},
 {4.675, 4.625, 5.078, 5.828, 6.714, 7.661, 8.588, 9.4, 9.761,
  9.26, 7.973, 6.455, 5.133, 4.127, 3.339, 2.562, 1.884, 1.37,
  1.036, 0.8316, 0.7014, 0.6314, 0.6329, 0.6673, 0.7434, 1.216, 1.142,
  1.346, 1.515, 1.614, 1.773, 2.186, 2.758, 3.13},
 {4.664, 4.649, 5.144, 5.923, 6.826, 7.78, 8.69, 9.369, 9.701,
  9.297, 8.02, 6.479, 5.147, 4.209, 3.474, 2.764, 2.093, 1.519,
  1.113, 0.8486, 0.683, 0.5849, 0.5324, 0.5209, 0.5327, 1.058, 0.9241,
  1.048, 1.287, 1.427, 1.546, 1.782, 2.111, 2.301},
 {4.771, 4.822, 5.408, 6.232, 7.155, 8.095, 8.92, 9.55, 9.74,
  9.14, 7.776, 6.287, 5.065, 4.198, 3.518, 2.824, 2.145, 1.537,
  1.096, 0.8121, 0.644, 0.5446, 0.4895, 0.4683, 0.463, 0.4501, 0.773,
  0.7994, 0.9657, 1.138, 1.222, 1.365, 1.564, 1.666},
 {5.238, 5.493, 6.287, 7.287, 8.326, 9.282, 10.02, 10.41, 10.22,
  9.194, 7.64, 6.172, 5.058, 4.272, 3.57, 2.766, 1.962, 1.313,
  0.8882, 0.6342, 0.4953, 0.4159, 0.3845, 0.3668, 0.3524, 0.3383, 0.6226,
  0.6524, 0.702, 0.662, 0.6453, 0.6304, 0.6111, 0.5462},
 {5.459, 5.786, 6.717, 8.018, 9.426, 10.72, 11.67, 11.98, 11.43,
  9.923, 8.082, 6.51, 5.425, 4.719, 3.958, 2.937, 1.953, 1.26,
  0.8432, 0.5868, 0.4389, 0.3576, 0.3281, 0.3182, 0.296, 0.2263, 0.5208,
  0.5264, 0.5272, 0.439, 0.4076, 0.3774, 0.3392, 0.2954}},
{{1.93, 2.082, 2.236, 2.401, 2.486, 2.46, 2.242, 1.936, 1.632,
  1.309, 1.205, 0.996, 0.8843, 0.5832, 0.3788, 0.2472, 0.1935, 0.199,
  0.2177, 0.3668, 0.2468, 0.1727, 0.1235, 0.1211, 0.09577, 0.05738, 0.01593,
  0.01572, 0.01585, 0.01477, 0.01213, 0.01077, 0.008684, 0.005934},
 {2.406, 2.637, 3.006, 3.467, 3.941, 4.339, 4.464, 4.221, 3.692,
  2.961, 2.333, 1.856, 1.579, 1.321, 0.9877, 0.7954, 0.535, 0.3953,
  0.3269, 0.3153, 0.4016, 0.4948, 0.4946, 0.3969, 0.2986, 0.157, 0.08327,
  0.09294, 0.1047, 0.09143, 0.08129, 0.06982, 0.05108, 0.0324},
 {2.891, 3.082, 3.531, 4.114, 4.759, 5.387, 5.81, 5.856, 5.447,
  4.602, 3.716, 2.967, 2.422, 1.997, 1.543, 1.312, 1.043, 0.7202,
  0.5009, 0.3828, 0.3369, 0.3204, 0.3053, 0.2956, 0.2344, 0.3256, 0.2033,
  0.2183, 0.2574, 0.252, 0.2454, 0.2496, 0.2494, 0.2289},
 {3.257, 3.412, 3.896, 4.558, 5.307, 6.077, 6.732, 7.047, 6.849,
  6.037, 4.935, 3.973, 3.242, 2.755, 2.32, 1.807, 1.313, 0.9215,
  0.6619, 0.4999, 0.3902, 0.3134, 0.2686, 0.245, 0.2368, 0.2048, 0.1813,
  0.2732, 0.3298, 0.3568, 0.3526, 0.3493, 0.3549, 0.3469},
 {3.587, 3.682, 4.173, 4.839, 5.622, 6.472, 7.24, 7.722, 7.706,
  6.99, 5.826, 4.706, 3.861, 3.255, 2.675, 1.997, 1.417, 0.9866,
  0.7187, 0.561, 0.4546, 0.3832, 0.3527, 0.3395, 0.3279, 0.4213, 0.3649,
  0.4532, 0.5745, 0.6214, 0.6234, 0.6274, 0.6214, 0.5786},
 {3.86, 3.88, 4.313, 4.965, 5.741, 6.605, 7.439, 8.066, 8.231,
  7.669, 6.514, 5.29, 4.308, 3.595, 2.953, 2.22, 1.613, 1.142,
  0.8371, 0.6574, 0.5366, 0.4574, 0.4295, 0.4016, 0.3794, 0.5616, 0.5829,
  0.7168, 0.9521, 1.025, 1.053, 1.129, 1.25, 1.317},
 {4.081, 4.041, 4.427, 5.032, 5.788, 6.668, 7.53, 8.225, 8.513,
  8.083, 6.997, 5.729, 4.642, 3.826, 3.135, 2.369, 1.741, 1.266,
  0.9467, 0.7487, 0.6194, 0.5505, 0.6097, 0.7323, 0.5351, 0.6646, 0.7098,
  0.9842, 1.318, 1.49, 1.534, 1.718, 2.035, 2.285},
 {4.216, 4.126, 4.503, 5.084, 5.827, 6.712, 7.64, 8.42, 8.793,
  8.425, 7.34, 6.028, 4.838, 3.895, 3.074, 2.264, 1.681, 1.272,
  1.009, 0.8308, 0.7345, 0.7299, 0.8734, 1.142, 0.5401, 0.7387, 0.9717,
  1.328, 1.652, 1.667, 1.69, 1.954, 2.622, 3.19},
 {4.272, 4.151, 4.513, 5.089, 5.851, 6.773, 7.683, 8.504, 8.944,
  8.636, 7.572, 6.215, 4.931, 3.883, 2.974, 2.147, 1.591, 1.231,
  1.009, 0.859, 0.7919, 0.8024, 0.896, 0.4226, 0.5516, 0.8571, 1.354,
  1.883, 1.918, 1.778, 1.904, 2.264, 2.865, 3.19},
 {4.268, 4.149, 4.512, 5.115, 5.895, 6.826, 7.773, 8.616, 9.068,
  8.77, 7.691, 6.307, 4.976, 3.859, 2.932, 2.123, 1.578, 1.222,
  0.9995, 0.8488, 0.7895, 0.8146, 0.9335, 0.4125, 0.6071, 0.932, 1.493,
  1.986, 1.896, 1.55, 1.538, 1.583, 1.731, 1.792},
 {4.24, 4.152, 4.565, 5.21, 6.012, 6.947, 7.876, 8.687, 9.116,
  8.769, 7.647, 6.266, 4.953, 3.871, 2.972, 2.151, 1.594, 1.22,
  0.9831, 0.8291, 0.7694, 0.7899, 0.9141, 0.4431, 0.6319, 0.9495, 1.544,
  2.116, 2.045, 1.603, 1.547, 1.538, 1.698, 1.778},
 {4.22, 4.152, 4.587, 5.265, 6.061, 6.961, 7.879, 8.672, 9.045,
```

```
   8.614, 7.444, 6.083, 4.845, 3.844, 2.996, 2.197, 1.608, 1.21,
   0.9591, 0.8021, 0.7294, 0.7305, 0.8376, 1.102, 0.495, 0.808, 1.134,
   1.49, 1.689, 1.458, 1.475, 1.619, 1.978, 2.214},
  {4.139, 4.115, 4.593, 5.262, 6.061, 6.95, 7.815, 8.557, 8.84,
   8.311, 7.1, 5.777, 4.638, 3.742, 2.972, 2.201, 1.594, 1.167,
   0.9052, 0.7436, 0.6504, 0.62, 0.7047, 0.8275, 0.483, 0.7578, 0.9831,
   1.246, 1.432, 1.377, 1.433, 1.677, 2.116, 2.395},
  {4.027, 4.03, 4.521, 5.195, 5.986, 6.839, 7.642, 8.241, 8.387,
   7.78, 6.581, 5.331, 4.311, 3.538, 2.878, 2.173, 1.57, 1.118,
   0.8368, 0.6691, 0.5655, 0.5116, 0.5213, 0.5546, 0.5962, 0.7812, 0.7265,
   0.8823, 1.107, 1.292, 1.385, 1.633, 2.016, 2.214},
  {3.787, 3.852, 4.369, 5.07, 5.845, 6.625, 7.28, 7.735, 7.755,
   7.082, 5.9, 4.755, 3.857, 3.202, 2.64, 2.031, 1.478, 1.036,
   0.7546, 0.5863, 0.4836, 0.422, 0.3872, 0.3827, 0.4058, 0.6138, 0.5067,
   0.6048, 0.7857, 0.9735, 1.052, 1.172, 1.322, 1.358},
  {3.556, 3.717, 4.24, 4.935, 5.651, 6.309, 6.815, 7.1, 6.959,
   6.199, 5.099, 4.12, 3.39, 2.875, 2.37, 1.798, 1.291, 0.8979,
   0.6433, 0.4909, 0.3991, 0.3437, 0.3063, 0.2936, 0.2983, 0.283, 0.3393,
   0.3825, 0.4825, 0.6185, 0.6697, 0.7099, 0.7581, 0.7492},
  {3.344, 3.611, 4.169, 4.812, 5.45, 5.983, 6.257, 6.263, 5.844,
   4.989, 4.049, 3.337, 2.825, 2.403, 1.893, 1.346, 0.9248, 0.6385,
   0.4618, 0.3493, 0.281, 0.2397, 0.2114, 0.1989, 0.1936, 0.1739, 0.1941,
   0.2137, 0.2382, 0.2613, 0.2565, 0.2526, 0.2392, 0.2172},
  {3.617, 4.108, 4.785, 5.409, 5.873, 6.058, 5.747, 5.269, 4.577,
   3.776, 3.142, 2.721, 2.304, 1.764, 1.195, 0.7421, 0.4648, 0.3052,
   0.2132, 0.1552, 0.1201, 0.1028, 0.09496, 0.09272, 0.092, 0.06578, 0.09663,
   0.1039, 0.1161, 0.1127, 0.1081, 0.1019, 0.09228, 0.08144}},
 {{4.776, 5.263, 6.105, 7.209, 8.284, 9.138, 9.553, 9.22, 8.161,
   6.644, 5.129, 4.002, 3.207, 2.842, 2.466, 1.792, 1.171, 0.7381,
   0.5022, 0.4253, 0.4401, 0.5454, 0.7785, 0.6418, 0.4899, 0.3887, 0.3859,
   0.2463, 0.248, 0.1831, 0.1414, 0.1233, 0.104, 0.07015},
  {4.315, 4.602, 5.253, 6.073, 6.933, 7.732, 8.306, 8.428, 7.938,
   6.819, 5.489, 4.358, 3.446, 2.895, 2.476, 1.885, 1.473, 1.07,
   0.7373, 0.5219, 0.4618, 0.454, 0.4569, 0.4263, 0.397, 0.4683, 0.4728,
   0.4348, 0.4638, 0.3711, 0.3221, 0.2786, 0.2347, 0.153},
  {4.034, 4.197, 4.772, 5.576, 6.46, 7.358, 8.131, 8.568, 8.457,
   7.592, 6.31, 5.095, 4.065, 3.363, 2.711, 2.026, 1.686, 1.33,
   1.006, 0.7144, 0.5379, 0.4588, 0.4427, 0.451, 0.4738, 0.5938, 0.595,
   0.5555, 0.6354, 0.6081, 0.5877, 0.5953, 0.6217, 0.6394},
  {4.052, 4.154, 4.699, 5.472, 6.377, 7.353, 8.242, 8.879, 9.016,
   8.382, 7.119, 5.777, 4.673, 3.861, 3.197, 2.408, 1.717, 1.19,
   0.8643, 0.6598, 0.5193, 0.4314, 0.3937, 0.3863, 0.4011, 0.3591, 0.4105,
   0.4673, 0.5539, 0.5717, 0.5531, 0.5518, 0.5563, 0.5379},
  {4.153, 4.213, 4.735, 5.466, 6.334, 7.3, 8.25, 8.998, 9.286,
   8.779, 7.558, 6.165, 5, 4.154, 3.419, 2.575, 1.849, 1.298,
   0.9519, 0.7439, 0.6103, 0.537, 0.545, 0.5816, 0.5683, 0.7049, 0.6064,
   0.7504, 0.9181, 0.9021, 0.8857, 0.8707, 0.8466, 0.7728},
  {4.248, 4.254, 4.738, 5.442, 6.269, 7.19, 8.14, 8.939, 9.34,
   8.978, 7.82, 6.411, 5.184, 4.273, 3.525, 2.664, 1.942, 1.394,
   1.033, 0.8143, 0.6732, 0.5954, 0.6154, 0.6352, 0.7198, 0.8339, 0.8119,
   1.046, 1.311, 1.244, 1.264, 1.359, 1.508, 1.629},
  {4.354, 4.315, 4.761, 5.42, 6.219, 7.126, 8.036, 8.844, 9.254,
   8.926, 7.833, 6.444, 5.174, 4.22, 3.457, 2.639, 1.948, 1.432,
   1.087, 0.8685, 0.7349, 0.6908, 0.8, 0.9447, 0.6798, 0.8977, 1.108,
   1.425, 1.666, 1.47, 1.449, 1.545, 1.768, 1.943},
  {4.362, 4.274, 4.672, 5.312, 6.096, 7.012, 7.964, 8.785, 9.19,
   8.837, 7.774, 6.356, 5.026, 3.988, 3.137, 2.353, 1.747, 1.332,
   1.065, 0.8964, 0.8178, 0.823, 0.9551, 1.158, 0.5854, 0.9884, 1.483,
   2.035, 2.091, 1.603, 1.521, 1.55, 1.723, 1.854},
  {4.3, 4.174, 4.537, 5.167, 5.965, 6.89, 7.821, 8.663, 9.083,
   8.735, 7.634, 6.25, 4.936, 3.878, 2.975, 2.165, 1.611, 1.244,
   1.01, 0.864, 0.8216, 0.8541, 0.947, 0.4242, 0.4811, 0.9771, 1.569,
   2.131, 2.042, 1.616, 1.566, 1.597, 1.77, 1.859},
  {4.191, 4.082, 4.474, 5.086, 5.853, 6.754, 7.678, 8.491, 8.887,
   8.503, 7.408, 6.052, 4.775, 3.739, 2.846, 2.066, 1.531, 1.182,
   0.9574, 0.8217, 0.7847, 0.8129, 0.9004, 0.3845, 0.5912, 0.9488, 1.54,
   2.073, 1.987, 1.591, 1.544, 1.55, 1.68, 1.746},
  {4.067, 4, 4.384, 4.981, 5.736, 6.618, 7.486, 8.26, 8.602,
   8.169, 7.063, 5.748, 4.553, 3.579, 2.743, 1.997, 1.466, 1.117,
   0.8947, 0.756, 0.7085, 0.7246, 0.8226, 0.3804, 0.4137, 0.8378, 1.338,
   1.831, 1.823, 1.452, 1.396, 1.406, 1.546, 1.607},
  {3.852, 3.824, 4.228, 4.825, 5.559, 6.404, 7.233, 7.922, 8.179,
   7.678, 6.56, 5.348, 4.269, 3.383, 2.62, 1.89, 1.369, 1.029,
   0.8128, 0.6781, 0.613, 0.6018, 0.6705, 0.6956, 0.4126, 0.6338, 0.8946,
   1.186, 1.405, 1.278, 1.277, 1.398, 1.661, 1.83},
  {3.577, 3.6, 4.019, 4.632, 5.336, 6.114, 6.899, 7.485, 7.599,
   6.995, 5.898, 4.813, 3.907, 3.169, 2.502, 1.82, 1.298, 0.9403,
   0.7213, 0.5834, 0.4987, 0.4571, 0.4767, 0.4929, 0.357, 0.5342, 0.6218,
   0.8321, 1.072, 1.133, 1.182, 1.326, 1.548, 1.687},
  {3.251, 3.338, 3.79, 4.396, 5.094, 5.833, 6.465, 6.856, 6.763,
   6.045, 5.018, 4.111, 3.394, 2.814, 2.265, 1.671, 1.175, 0.8195,
   0.6074, 0.4777, 0.3953, 0.3477, 0.3295, 0.3281, 0.3202, 0.3996, 0.4085,
   0.5264, 0.6945, 0.8446, 0.9062, 1.002, 1.145, 1.182},
  {2.844, 3.014, 3.474, 4.053, 4.689, 5.317, 5.776, 5.907, 5.606,
   4.864, 3.98, 3.263, 2.726, 2.292, 1.824, 1.308, 0.8985, 0.62,
   0.4532, 0.3508, 0.2879, 0.2496, 0.2268, 0.217, 0.2089, 0.1723, 0.2384,
   0.3033, 0.3816, 0.457, 0.4738, 0.4945, 0.522, 0.511},
```

```
    {2.32, 2.558, 2.981, 3.486, 3.981, 4.383, 4.505, 4.35, 3.92,
     3.298, 2.688, 2.238, 1.901, 1.574, 1.197, 0.8142, 0.5528, 0.3893,
     0.2874, 0.2207, 0.179, 0.153, 0.1353, 0.126, 0.118, 0.0996, 0.1038,
     0.1254, 0.1556, 0.1673, 0.1637, 0.1606, 0.1599, 0.1512},
    {1.604, 1.812, 2.085, 2.33, 2.517, 2.565, 2.341, 2.07, 1.768,
     1.479, 1.254, 1.072, 0.8657, 0.6304, 0.4269, 0.2773, 0.1887, 0.1365,
     0.104, 0.08106, 0.06679, 0.05742, 0.0504, 0.04538, 0.04064, 0.03211,
     0.02611,
     0.02727, 0.03066, 0.03125, 0.02913, 0.02667, 0.02485, 0.02217},
    {0.4429, 0.4652, 0.4579, 0.4357, 0.388, 0.3401, 0.2901, 0.2551, 0.2192,
     0.1788, 0.1348, 0.0914, 0.05502, 0.03148, 0.01858, 0.01216, 0.009078,
     0.007534,
     0.006492, 0.006257, 0.006301, 0.006115, 0.005378, 0.005084, 0.005022,
     0.002766, 0.00246,
     0.001431, 0.001208, 0.0014, 0.001257, 0.001091, 0.0009076, 0.0005632}},
   {{6.159, 6.267, 7.137, 8.441, 9.868, 11.32, 12.68, 13.44, 13.31,
     11.97, 9.899, 7.982, 6.529, 5.456, 4.582, 3.411, 2.447, 1.735,
     1.201, 0.8902, 0.7385, 0.82, 0.8247, 0.6792, 0.5978, 0.7594, 1.311,
     0.8001, 0.7222, 0.5196, 0.4, 0.3513, 0.3049, 0.2033},
    {6.025, 6.218, 7.108, 8.305, 9.599, 10.86, 11.89, 12.5, 12.4,
     11.24, 9.372, 7.562, 6.173, 5.198, 4.359, 3.187, 2.293, 1.674,
     1.209, 0.8758, 0.6848, 0.7086, 0.7544, 0.6892, 0.6462, 0.8934, 1.25,
     0.9268, 0.9078, 0.7087, 0.6172, 0.5399, 0.4671, 0.304},
    {5.423, 5.508, 6.112, 6.997, 8.017, 9.107, 10.07, 10.75, 10.95,
     10.26, 8.775, 7.139, 5.746, 4.723, 3.902, 2.898, 2.14, 1.579,
     1.155, 0.86, 0.6747, 0.5805, 0.5689, 0.5958, 0.6918, 0.8863, 1.028,
     0.8369, 0.8899, 0.8223, 0.7867, 0.7493, 0.7297, 0.6911},
    {5.03, 5.007, 5.533, 6.337, 7.269, 8.306, 9.353, 10.16, 10.53,
     10.09, 8.802, 7.176, 5.772, 4.785, 3.999, 3.116, 2.292, 1.613,
     1.161, 0.8825, 0.7044, 0.602, 0.5675, 0.5799, 0.633, 0.5335, 0.75,
     0.782, 0.8887, 0.8566, 0.8053, 0.779, 0.7512, 0.6894},
    {4.854, 4.795, 5.239, 5.991, 6.89, 7.916, 8.949, 9.794, 10.25,
     9.932, 8.787, 7.214, 5.806, 4.824, 4.046, 3.151, 2.318, 1.656,
     1.216, 0.9337, 0.7505, 0.6429, 0.6093, 0.6241, 0.6313, 0.828, 0.8414,
     1.007, 1.191, 1.083, 1.037, 1.002, 0.963, 0.8665},
    {4.689, 4.616, 5.08, 5.801, 6.661, 7.626, 8.619, 9.471, 9.972,
     9.711, 8.582, 7.039, 5.65, 4.664, 3.923, 3.04, 2.236, 1.63,
     1.223, 0.9546, 0.7831, 0.6819, 0.6534, 0.6599, 0.5823, 0.9021, 1.016,
     1.267, 1.495, 1.395, 1.398, 1.508, 1.711, 1.859},
    {4.575, 4.466, 4.925, 5.634, 6.479, 7.412, 8.343, 9.218, 9.74,
     9.468, 8.319, 6.801, 5.4, 4.37, 3.571, 2.713, 2.014, 1.498,
     1.158, 0.9364, 0.7954, 0.7273, 0.7681, 0.8566, 0.6687, 0.9463, 1.219,
     1.575, 1.788, 1.546, 1.525, 1.635, 1.887, 2.09},
    {4.408, 4.299, 4.735, 5.433, 6.275, 7.22, 8.175, 9.032, 9.47,
     9.097, 7.942, 6.45, 5.08, 4.019, 3.16, 2.338, 1.735, 1.324,
     1.063, 0.9095, 0.8421, 0.8249, 0.8269, 0.6041, 0.6079, 1.07, 1.601,
     2.169, 2.204, 1.68, 1.593, 1.624, 1.805, 1.939},
    {4.26, 4.136, 4.516, 5.172, 5.992, 6.946, 7.925, 8.739, 9.102,
     8.667, 7.534, 6.13, 4.865, 3.837, 2.924, 2.126, 1.572, 1.207,
     0.9693, 0.8391, 0.8016, 0.8213, 0.8584, 0.4114, 0.632, 0.9986, 1.59,
     2.132, 2.045, 1.617, 1.567, 1.595, 1.754, 1.846},
    {4.068, 3.944, 4.321, 4.953, 5.759, 6.696, 7.598, 8.349, 8.636,
     8.134, 6.966, 5.681, 4.531, 3.592, 2.726, 1.981, 1.454, 1.112,
     0.8863, 0.7642, 0.7336, 0.7503, 0.7883, 0.3649, 0.5742, 0.9245, 1.482,
     1.979, 1.91, 1.537, 1.492, 1.497, 1.609, 1.655},
    {3.784, 3.702, 4.08, 4.686, 5.436, 6.274, 7.107, 7.772, 7.978,
     7.457, 6.38, 5.207, 4.171, 3.317, 2.572, 1.856, 1.347, 1.012,
     0.8031, 0.6877, 0.6459, 0.6441, 0.6934, 0.3392, 0.5146, 0.7711, 1.206,
     1.627, 1.668, 1.358, 1.31, 1.315, 1.411, 1.432},
    {3.39, 3.409, 3.832, 4.448, 5.159, 5.932, 6.66, 7.149, 7.206,
     6.618, 5.602, 4.608, 3.743, 3.01, 2.347, 1.689, 1.206, 0.8923,
     0.6966, 0.5763, 0.5136, 0.4878, 0.5216, 0.5783, 0.3499, 0.515, 0.7012,
     0.9131, 1.167, 1.133, 1.139, 1.212, 1.359, 1.445},
    {3.031, 3.122, 3.551, 4.115, 4.781, 5.496, 6.101, 6.433, 6.32,
     5.654, 4.707, 3.886, 3.211, 2.629, 2.053, 1.473, 1.024, 0.7318,
     0.5579, 0.445, 0.3748, 0.3356, 0.3272, 0.3261, 0.3502, 0.4067, 0.4482,
     0.5625, 0.7534, 0.8328, 0.8615, 0.9261, 1.038, 1.075},
    {2.556, 2.697, 3.11, 3.64, 4.251, 4.887, 5.363, 5.492, 5.176,
     4.453, 3.662, 3.064, 2.599, 2.164, 1.677, 1.161, 0.7816, 0.5445,
     0.4076, 0.3171, 0.258, 0.2227, 0.2043, 0.1946, 0.1903, 0.2423, 0.2411,
     0.2984, 0.3661, 0.4305, 0.4483, 0.4735, 0.5096, 0.5082},
    {1.982, 2.163, 2.522, 2.962, 3.444, 3.894, 4.12, 3.996, 3.538,
     2.915, 2.39, 2.044, 1.761, 1.418, 1.026, 0.6684, 0.4452, 0.3147,
     0.2354, 0.1814, 0.1474, 0.1272, 0.1136, 0.1042, 0.09334, 0.07244, 0.09453,
     0.1067, 0.1323, 0.1309, 0.1255, 0.1235, 0.1251, 0.1207},
    {1.313, 1.48, 1.706, 1.932, 2.113, 2.193, 2.081, 1.804, 1.487,
     1.196, 0.9808, 0.8365, 0.6791, 0.4931, 0.3304, 0.2112, 0.1439, 0.1054,
     0.08052, 0.06314, 0.05248, 0.04667, 0.0419, 0.03731, 0.03192, 0.02135,
     0.01682,
     0.0156, 0.01767, 0.01723, 0.0161, 0.01526, 0.0148, 0.01411},
    {0.242, 0.2311, 0.2162, 0.1962, 0.1752, 0.1604, 0.1387, 0.1112, 0.08183,
     0.05815, 0.04045, 0.02676, 0.01677, 0.01075, 0.007653, 0.005984, 0.00512,
     0.004795,
     0.004786, 0.004999, 0.004952, 0.004352, 0.003443, 0.002664, 0.002223,
     0.001163, 0.001542,
     0.0002821, 0.0001951, 0.000206, 0.0001656, 0.0001206, 8.303e-05,
     5.901e-05},
```

```
 {0.0001232, 0.0001559, 0.0001539, 0.0001693, 0.0002134, 0.0002031,
  0.0001037, 1.126e-05, 5.382e-06,
  1.867e-06, 5.983e-07, 2.464e-07, 1.576e-07, 1.322e-07, 1.312e-07,
  1.319e-07, 3.921e-07, 3.583e-06,
  3.815e-05, 6.754e-05, 0.0001004, 0.0002135, 0.0004217, 0.0007681,
  0.001524, 0.0004274, 0.000876,
  2.698e-05, 1.328e-12, 1.445e-13, 9.798e-14, 8.583e-14, 9.786e-14,
  1.774e-13}},
{{6.595, 6.532, 7.313, 8.453, 9.864, 11.47, 13.06, 14.28, 14.67,
  13.68, 11.56, 9.275, 7.452, 6.201, 5.275, 4.16, 2.898, 2.003,
  1.4, 1.04, 0.7754, 0.7071, 0.7598, 0.799, 0.825, 0.9217, 1.851,
  1.254, 1.138, 0.8159, 0.6311, 0.5427, 0.4614, 0.3814},
 {6.516, 6.556, 7.327, 8.526, 9.924, 11.42, 12.85, 13.82, 14.03,
  13.05, 11.03, 8.863, 7.108, 5.878, 4.956, 3.797, 2.704, 1.92,
  1.344, 0.9685, 0.7276, 0.6364, 0.6746, 0.7239, 0.786, 0.9333, 1.793,
  1.344, 1.234, 0.8885, 0.7949, 0.6932, 0.5878, 0.4871},
 {6.179, 6.202, 6.853, 7.807, 8.924, 10.13, 11.21, 12.01, 12.29,
  11.63, 10.05, 8.152, 6.536, 5.386, 4.503, 3.473, 2.521, 1.809,
  1.273, 0.9058, 0.6837, 0.5774, 0.5746, 0.6269, 0.7726, 0.9434, 1.275,
  1.102, 1.148, 0.9922, 0.9195, 0.8713, 0.8162, 0.7358},
 {5.401, 5.302, 5.812, 6.634, 7.64, 8.785, 9.902, 10.82, 11.3,
  10.96, 9.696, 7.981, 6.412, 5.281, 4.41, 3.469, 2.606, 1.892,
  1.37, 1.034, 0.8087, 0.6766, 0.6565, 0.6981, 0.7901, 0.6904, 1.01,
  1.062, 1.192, 1.063, 1.016, 0.9639, 0.8911, 0.7914},
 {5.101, 4.973, 5.426, 6.18, 7.138, 8.24, 9.32, 10.29, 10.9,
  10.75, 9.665, 8.035, 6.469, 5.319, 4.452, 3.502, 2.649, 1.941,
  1.431, 1.09, 0.869, 0.7456, 0.7339, 0.7833, 0.8079, 1.059, 1.104,
  1.303, 1.515, 1.253, 1.185, 1.131, 1.076, 0.9437},
 {4.936, 4.795, 5.272, 5.985, 6.878, 7.91, 8.989, 9.922, 10.53,
  10.37, 9.278, 7.698, 6.176, 5.044, 4.178, 3.263, 2.472, 1.849,
  1.402, 1.087, 0.8859, 0.7846, 0.8226, 0.8854, 0.9635, 1.037, 1.251,
  1.527, 1.706, 1.5, 1.503, 1.644, 1.914, 2.113},
 {4.796, 4.617, 5.024, 5.703, 6.591, 7.617, 8.632, 9.544, 10.07,
  9.749, 8.552, 6.983, 5.55, 4.462, 3.573, 2.707, 2.021, 1.537,
  1.216, 1.017, 0.9039, 0.8702, 0.9836, 1.21, 0.6125, 1.009, 1.311,
  1.688, 1.862, 1.575, 1.568, 1.696, 2.001, 2.214},
 {4.522, 4.356, 4.742, 5.465, 6.36, 7.357, 8.359, 9.269, 9.706,
  9.237, 7.95, 6.476, 5.137, 4.086, 3.214, 2.373, 1.75, 1.33,
  1.071, 0.9379, 0.8929, 0.9071, 0.9736, 1.305, 0.5218, 1.054, 1.605,
  2.105, 1.976, 1.563, 1.521, 1.56, 1.765, 1.875},
 {4.201, 4.084, 4.453, 5.134, 5.998, 7.007, 8.042, 8.894, 9.218,
  8.665, 7.393, 5.966, 4.728, 3.77, 2.956, 2.16, 1.585, 1.199,
  0.9637, 0.8579, 0.8414, 0.8686, 0.8189, 1.154, 0.4693, 0.9934, 1.568,
  2.075, 1.962, 1.563, 1.524, 1.545, 1.704, 1.786},
 {3.87, 3.761, 4.135, 4.74, 5.547, 6.523, 7.533, 8.287, 8.542,
  7.978, 6.743, 5.463, 4.36, 3.491, 2.739, 1.993, 1.453, 1.095,
  0.8767, 0.7822, 0.7664, 0.777, 0.8145, 1.109, 0.4094, 0.8854, 1.413,
  1.91, 1.872, 1.47, 1.421, 1.428, 1.538, 1.583},
 {3.565, 3.517, 3.908, 4.525, 5.299, 6.159, 6.982, 7.581, 7.734,
  7.15, 6.028, 4.918, 3.993, 3.242, 2.541, 1.833, 1.321, 0.9862,
  0.7851, 0.6877, 0.6504, 0.6409, 0.6657, 0.7916, 0.3852, 0.627, 0.8774,
  1.306, 1.713, 1.397, 1.317, 1.308, 1.379, 1.377},
 {3.27, 3.307, 3.718, 4.324, 5.008, 5.72, 6.391, 6.82, 6.844,
  6.25, 5.256, 4.321, 3.562, 2.929, 2.309, 1.67, 1.183, 0.8581,
  0.6613, 0.5437, 0.4817, 0.4549, 0.4828, 0.4971, 0.343, 0.4517, 0.5928,
  0.7482, 1.114, 1.156, 1.127, 1.142, 1.266, 1.325},
 {2.881, 2.972, 3.365, 3.885, 4.479, 5.095, 5.612, 5.869, 5.739,
  5.109, 4.233, 3.497, 2.928, 2.45, 1.923, 1.37, 0.937, 0.6588,
  0.4974, 0.3913, 0.3216, 0.2799, 0.263, 0.2476, 0.2702, 0.3664, 0.3897,
  0.4754, 0.6181, 0.6968, 0.7144, 0.7507, 0.8199, 0.8256},
 {2.352, 2.522, 2.914, 3.377, 3.888, 4.391, 4.73, 4.773, 4.456,
  3.814, 3.103, 2.576, 2.19, 1.824, 1.372, 0.9129, 0.606, 0.4281,
  0.3241, 0.25, 0.1992, 0.1685, 0.1489, 0.1316, 0.116, 0.1598, 0.1448,
  0.1805, 0.2224, 0.2379, 0.2369, 0.2454, 0.2679, 0.2718},
 {1.666, 1.833, 2.135, 2.486, 2.847, 3.14, 3.202, 3.006, 2.612,
  2.127, 1.726, 1.486, 1.277, 0.9733, 0.6654, 0.4233, 0.2852, 0.2051,
  0.1537, 0.1174, 0.09422, 0.08017, 0.06975, 0.06009, 0.04775, 0.03319,
  0.03371,
  0.03896, 0.04544, 0.04203, 0.03927, 0.03814, 0.03917, 0.04012},
 {0.8975, 0.9719, 1.03, 1.066, 1.034, 0.9374, 0.7957, 0.662, 0.5656,
  0.4856, 0.4141, 0.3239, 0.2283, 0.1478, 0.09439, 0.06056, 0.04188,
  0.03179,
  0.02625, 0.02293, 0.02134, 0.01999, 0.01796, 0.01508, 0.01136, 0.006243,
  0.005399,
  0.002554, 0.002671, 0.002877, 0.002693, 0.002456, 0.002169, 0.001592},
 {0.005568, 0.003081, 0.001936, 0.001388, 0.001138, 0.0009141, 0.0003913,
  7.042e-05, 1.305e-05,
  9.014e-06, 5.819e-06, 3.047e-06, 1.303e-06, 5.602e-07, 2.183e-07,
  1.757e-07, 3.825e-07, 2.566e-06,
  1.334e-05, 1.436e-05, 1.976e-05, 7.261e-05, 0.0002657, 0.0005962,
  0.001653, 0.0002773, 0.0008521,
  1.309e-06, 3.72e-14, 2.315e-16, 2.404e-15, 7.283e-17, 5.816e-17,
  1.165e-16},
 {5.606e-05, 7.174e-05, 7.065e-05, 8.779e-05, 0.0001175, 0.0001418,
  6.181e-05, 7.462e-06, 8.135e-06,
  6.922e-06, 3.21e-06, 1.063e-06, 3.185e-07, 7.307e-08, 1.298e-08,
  1.751e-08, 6.792e-08, 5.277e-07,
```

```
    7.612e-06, 1.832e-05, 4.78e-05, 0.0001019, 0.0001703, 0.0003801, 0.001213,
    0.0002105, 0.0006011,
    2.875e-06, 7.798e-13, 1.214e-13, 8.329e-14, 7.553e-14, 1.014e-13,
    1.901e-13}}
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_oh_var)
#endif

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

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Check pressure... */
  if (p < clim_oh_ps[0])
    p = clim_oh_ps[0];
  else if (p > clim_oh_ps[33])
    p = clim_oh_ps[33];

  /* Get indices... */
  int isec = locate_irr(clim_oh_secs, 12, sec);
  int ilat = locate_reg(clim_oh_lats, 18, lat);
  int ip = locate_irr(clim_oh_ps, 34, p);

  /* Interpolate OH climatology (Pommrich et al., 2014)... */
  double aux00 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec][ilat][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec][ilat + 1][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec + 1][ilat][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec + 1][ilat + 1][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
  aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
  return 1e6 * LIN(clim_oh_secs[isec], aux00,
                   clim_oh_secs[isec + 1], aux11, sec);
}

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

static double clim_tropo_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
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_tropo_secs)
#endif

static double clim_tropo_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
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_tropo_lats)
#endif

static double clim_tropo_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}
};

#ifdef _OPENACC
#pragma acc declare copyin(clim_tropo_tps)
#endif

double clim_tropo(
  double t,
  double lat) {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Get indices... */
  int isec = locate_irr(clim_tropo_secs, 12, sec);
  int ilat = locate_reg(clim_tropo_lats, 73, lat);

  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
  double p0 = LIN(clim_tropo_lats[ilat],
                  clim_tropo_tps[isec][ilat],
                  clim_tropo_lats[ilat + 1],
                  clim_tropo_tps[isec][ilat + 1], lat);
  double p1 = LIN(clim_tropo_lats[ilat],
                  clim_tropo_tps[isec + 1][ilat],
                  clim_tropo_lats[ilat + 1],
                  clim_tropo_tps[isec + 1][ilat + 1], lat);
  return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
}

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

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 (d0l[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0l[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 = 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!");
#ifdef _OPENACC
    met_t *met0up = *met0;
    met_t *met1up = *met1;
#pragma acc update device(met0up[:1],met1up[:1])
#endif
  }

  /* 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!");
#ifdef _OPENACC
    met_t *met1up = *met1;
#pragma acc update device(met1up[:1])
#endif
  }

  /* 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!");
#ifdef _OPENACC
    met_t *met0up = *met0;
#pragma acc update device(met0up[:1])
#endif
  }

  /* 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;

  /* Iterate... */
  for (int 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_space_3d(
  met_t * met,
  float array[EX][EY][EP],
  double p,
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init) {

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

  /* Get interpolation indices and weights... */
  if (init) {
    ci[0] = locate_irr(met->p, met->np, p);
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);
    cw[0] = (met->p[ci[0] + 1] - p)
      / (met->p[ci[0] + 1] - met->p[ci[0]]);
    cw[1] = (met->lon[ci[1] + 1] - lon)
      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
    cw[2] = (met->lat[ci[2] + 1] - lat)
      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
  }
```

```

  /* Interpolate vertically... */
  double aux00 =
    cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
    + array[ci[1]][ci[2]][ci[0] + 1];
  double aux01 =
    cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
             array[ci[1]][ci[2] + 1][ci[0] + 1])
    + array[ci[1]][ci[2] + 1][ci[0] + 1];
  double aux10 =
    cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
             array[ci[1] + 1][ci[2]][ci[0] + 1])
    + array[ci[1] + 1][ci[2]][ci[0] + 1];
  double aux11 =
    cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
    + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];

  /* Interpolate horizontally... */
  aux00 = cw[2] * (aux00 - aux01) + aux01;
  aux11 = cw[2] * (aux10 - aux11) + aux11;
  *var = cw[1] * (aux00 - aux11) + aux11;
}


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

void intpol_met_space_2d(
  met_t * met,
  float array[EX][EY],
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init) {

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

  /* Get interpolation indices and weights... */
  if (init) {
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);
    cw[1] = (met->lon[ci[1] + 1] - lon)
      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
    cw[2] = (met->lat[ci[2] + 1] - lat)
      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
  }

  /* Set variables... */
  double aux00 = array[ci[1]][ci[2]];
  double aux01 = array[ci[1]][ci[2] + 1];
  double aux10 = array[ci[1] + 1][ci[2]];
  double aux11 = array[ci[1] + 1][ci[2] + 1];

  /* Interpolate horizontally... */
  if (isfinite(aux00) && isfinite(aux01))
    aux00 = cw[2] * (aux00 - aux01) + aux01;
  else if (cw[2] < 0.5)
    aux00 = aux01;
  if (isfinite(aux10) && isfinite(aux11))
    aux11 = cw[2] * (aux10 - aux11) + aux11;
  else if (cw[2] > 0.5)
    aux11 = aux10;
  if (isfinite(aux00) && isfinite(aux11))
    *var = cw[1] * (aux00 - aux11) + aux11;
  else {
    if (cw[1] > 0.5)
      *var = aux00;
    else
      *var = aux11;
  }
}

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

void intpol_met_time_3d(
  met_t * met0,
  float array0[EX][EY][EP],
  met_t * met1,
  float array1[EX][EY][EP],
  double ts,
  double p,
  double lon,
  double lat,
```

```
  double *var,
  int *ci,
  double *cw,
  int init) {

  double var0, var1, wt;

  /* Spatial interpolation... */
  intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
  intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);

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

  /* Interpolate... */
  *var = wt * (var0 - var1) + var1;
}

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

void intpol_met_time_2d(
  met_t * met0,
  float array0[EX][EY],
  met_t * met1,
  float array1[EX][EY],
  double ts,
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init) {

  double var0, var1, wt;

  /* Spatial interpolation... */
  intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
  intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);

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

  /* Interpolate... */
  *var = wt * (var0 - var1) + var1;
}

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

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

  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;

  time_t 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 ilo = 0;
```

```
  int ihi = n - 1;
  int 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) {

  /* Calculate index... */
  int 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"))) {
      WARN("File not found!");
      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) {

  /* 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_lwc = -1;
  ctl->qnt_iwc = -1;
  ctl->qnt_pc = -1;
  ctl->qnt_hno3 = -1;
  ctl->qnt_oh = -1;
  ctl->qnt_rh = -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 (int 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], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
      ctl->qnt_o3 = iq;
      sprintf(ctl->qnt_unit[iq], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
      ctl->qnt_lwc = iq;
      sprintf(ctl->qnt_unit[iq], "kg/kg");
    } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
      ctl->qnt_iwc = iq;
      sprintf(ctl->qnt_unit[iq], "kg/kg");
    } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
      ctl->qnt_pc = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
      ctl->qnt_hno3 = iq;
      sprintf(ctl->qnt_unit[iq], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
      ctl->qnt_oh = iq;
      sprintf(ctl->qnt_unit[iq], "molec/cm^3");
    } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
      ctl->qnt_rh = iq;
      sprintf(ctl->qnt_unit[iq], "%%");
    } 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 (int 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, "3", NULL);
  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);

  /* Species parameters... */
  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
  if (strcmp(ctl->species, "SO2") == 0) {
    ctl->molmass = 64.066;
    ctl->oh_chem[0] = 3.3e-31;   /* (JPL Publication 15-10) */
    ctl->oh_chem[1] = 4.3;       /* (JPL Publication 15-10) */
    ctl->oh_chem[2] = 1.6e-12;   /* (JPL Publication 15-10) */
    ctl->oh_chem[3] = 0.0;       /* (JPL Publication 15-10) */
    ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
    ctl->wet_depo[1] = 0.62;    /* (FLEXPART v10.4) */
    ctl->wet_depo[2] = 1.3e-2;  /* (Sander, 2015) */
    ctl->wet_depo[3] = 2900.0;  /* (Sander, 2015) */
  } else {
    ctl->molmass =
      scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", 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);
    for (int ip = 0; ip < 4; ip++)
      ctl->oh_chem[ip] =
        scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
    for (int ip = 0; ip < 4; ip++)
      ctl->wet_depo[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO", ip, "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_stride =
    (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", 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];

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

  size_t np, nx, ny;

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

  /* 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);

  /* 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) {
      WARN("File not found!");
      return 0;
    }
  }

  /* 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");
    if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
      sprintf(levname, "plev");
      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... */
  if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
    ERRMSG("Cannot read temperature!");
  if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
    ERRMSG("Cannot read zonal wind!");
  if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
    ERRMSG("Cannot read meridional wind!");
  if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
    WARN("Cannot read vertical velocity");
  if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
    WARN("Cannot read specific humidity!");
  if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
    WARN("Cannot read ozone data!");
  if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
    WARN("Cannot read cloud liquid water content!");
  if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
    WARN("Cannot read cloud ice water content!");

  /* 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_3d(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);
    read_met_ml2pl(ctl, met, met->lwc);
    read_met_ml2pl(ctl, met, met->iwc);

    /* 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 data... */
  read_met_surface(ncid, met);

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

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

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

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

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

```

```
  /* Calculate cloud properties... */
  read_met_cloud(met);

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

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

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

void read_met_cloud(
  met_t * met) {

  int ix, iy, ip;

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

      /* Init... */
      met->pc[ix][iy] = GSL_NAN;
      met->cl[ix][iy] = 0;

      /* Loop over pressure levels... */
      for (ip = 0; ip < met->np - 1; ip++) {

        /* Check pressure... */
        if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
          continue;

        /* Get cloud top pressure ... */
        if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
          met->pc[ix][iy] = (float) met->p[ip + 1];

        /* Get cloud water... */
        met->cl[ix][iy] += (float)
          (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
                  + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
           * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
      }
    }
}

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

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 (!isfinite(met->t[ix][iy][ip0])
            || !isfinite(met->u[ix][iy][ip0])
            || !isfinite(met->v[ix][iy][ip0])
            || !isfinite(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];
        met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
        met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
      }
    }
}

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

void read_met_geopot(
  met_t * met) {

  const int dx = 6, dy = 4;

```

```
  static float help[EX][EY][EP];

  double logp[EP], ts, z0, cw[3];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];

  /* Calculate log pressure... */
  for (ip = 0; ip < met->np; ip++)
    logp[ip] = log(met->p[ip]);

  /* 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;

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

      /* Get surface height... */
      intpol_met_space_2d(met, met->zs, met->lon[ix], met->lat[iy], &z0, ci,
                          cw, 1);

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

      /* Get virtual temperature at the surface... */
      ts =
        LIN(met->p[ip0],
            TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
            met->p[ip0 + 1],
            TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[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 + TVIRT(met->t[ix][iy][ip0 + 1],
                                 met->h2o[ix][iy][ip0 + 1]))
                   * (log(met->ps[ix][iy]) - logp[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 *
                     (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
                      + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
                     * (logp[ip - 1] - logp[ip]));
    }

  /* Horizontal smoothing... */
#pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        n = 0;
        help[ix][iy][ip] = 0;
        for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - dy, 0);
               iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
            if (isfinite(met->z[ix3][iy2][ip])) {
              help[ix][iy][ip] += met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0)
          help[ix][iy][ip] /= (float) n;
        else
          help[ix][iy][ip] = GSL_NAN;
      }

  /* Copy 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++)
        met->z[ix][iy][ip] = help[ix][iy][ip];
}

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

```
int read_met_help_3d(
  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 0;

  /* Allocate... */
  ALLOC(help, float, EX * EY * EP);

  /* 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);

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

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

int read_met_help_2d(
  int ncid,
  char *varname,
  char *varname2,
  met_t * met,
  float dest[EX][EY],
  float scl) {

  float *help;

  int 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 0;

  /* Allocate... */
  ALLOC(help, float, EX * EY);

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

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

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

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

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

```
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) {

  /* 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)
  for (int iy = 0; iy < met->ny; iy++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->zs[met->nx - 1][iy] = met->zs[0][iy];
    for (int ip = 0; ip < met->np; 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->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
      met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
      met->iwc[met->nx - 1][iy][ip] = met->iwc[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 = 0.5 * (met->lat[iy1] + met->lat[iy0]);
      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));
      }
    }
  }

  /* Fix for polar regions... */
#pragma omp parallel for default(shared) private(ix,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (ip = 0; ip < met->np; ip++) {
      met->pv[ix][0][ip]
        = met->pv[ix][1][ip]
        = met->pv[ix][2][ip];
      met->pv[ix][met->ny - 1][ip]
        = met->pv[ix][met->ny - 2][ip]
        = met->pv[ix][met->ny - 3][ip];
    }
}
```

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

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->zs[ix][iy] = 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->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        help->lwc[ix][iy][ip] = 0;
        help->iwc[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 - abs(ix - ix2) / ctl->met_sx)
                * (float) (1.0 - abs(iy - iy2) / ctl->met_sy)
                * (float) (1.0 - abs(ip - ip2) / ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->zs[ix][iy] += w * met->zs[ix3][iy2];
              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->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
              help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->zs[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
        help->lwc[ix][iy][ip] /= wsum;
        help->iwc[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->zs[met->nx][met->ny] = help->zs[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl->met_dp) {
        met->p[met->np] = help->p[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->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->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
        met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

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

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

void read_met_surface(
  int ncid,
  met_t * met) {

  int ix, iy;

  /* Read surface pressure... */
  if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
    if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
      ERRMSG("Cannot not read surface pressure data!");
      for (ix = 0; ix < met->nx; ix++)
        for (iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) met->p[0];
    } else {
      for (iy = 0; iy < met->ny; iy++)
        for (ix = 0; ix < met->nx; ix++)
          met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
    }
  }

  /* Read geopotential height at the surface... */
  if (!read_met_help_2d
      (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
    if (!read_met_help_2d
        (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
      ERRMSG("Cannot read surface geopotential height!");
}

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

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

  double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
    th2[200], z[EP], z2[200];

  int found, ix, iy, iz, iz2;

  /* Get altitude and pressure profiles... */
  for (iz = 0; iz < met->np; iz++)
    z[iz] = Z(met->p[iz]);
  for (iz = 0; iz <= 190; 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) {
#pragma omp parallel for default(shared) private(ix,iy)
```

```
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
  }

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
    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];
        spline(z, t, met->np, z2, t2, 171);

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

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
    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];
        spline(z, t, met->np, z2, t2, 191);

        /* Find 1st tropopause... */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++) {
          found = 1;
          for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
            if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
              found = 0;
              break;
            }
          if (found) {
            if (iz > 0 && iz < 170)
              met->pt[ix][iy] = (float) p2[iz];
            break;
          }
        }

        /* Find 2nd tropopause... */
        if (ctl->met_tropo == 4) {
          met->pt[ix][iy] = GSL_NAN;
          for (; iz <= 170; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 170; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
                found = 0;
                break;
              }
            if (found) {
              if (iz > 0 && iz < 170)
                met->pt[ix][iy] = (float) p2[iz];
              break;
            }
          }
        }
      }
```

```
  }

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
    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];
        spline(z, pv, met->np, z2, pv2, 171);

        /* Interpolate potential temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
        spline(z, th, met->np, z2, th2, 171);

        /* Find dynamical tropopause 3.5 PVU + 380 K */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++)
          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
            if (iz > 0 && iz < 170)
              met->pt[ix][iy] = (float) p2[iz];
            break;
          }
      }
  }

  else
    ERRMSG("Cannot calculate tropopause!");
}

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

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 spline(
  double *x,
  double *y,
  int n,
  double *x2,
  double *y2,
  int n2) {

  gsl_interp_accel *acc;

  gsl_spline *s;

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);

  /* Interpolate temperature profile... */
  gsl_spline_init(s, x, y, (size_t) n);
  for (int i = 0; i < n2; i++)
    if (x2[i] <= x[0])
      y2[i] = y[0];
    else if (x2[i] >= x[n - 1])
      y2[i] = y[n - 1];
    else
      y2[i] = gsl_spline_eval(s, x2[i], acc);

  /* Free... */
  gsl_spline_free(s);
  gsl_interp_accel_free(acc);
}

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

double stddev(
  double *data,
  int n) {

  if (n <= 0)
    return 0;

  double avg = 0, rms = 0;

  for (int i = 0; i < n; ++i)
    avg += data[i];
  avg /= n;

  for (int i = 0; i < n; ++i)
    rms += SQR(data[i] - avg);

  return sqrt(rms / (n - 1));
}

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

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 += ctl->atm_stride) {

      /* 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, cw[3];

  static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
    ci[3];

  /* 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 [ppv]\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_3d(met0, met0->t, met1, met1->t, t, press, lon,
                             lat, &temp, ci, cw, 1);
```

```

          /* 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 = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;

          /* 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], rt, rz, rlon, rlat, robs,
    t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
    o3, cw[3];

  static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];

  /* 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!");

    /* Check molar mass... */
    if (ctl->molmass <= 0)
      ERRMSG("Specify molar mass!");

    /* 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 [ppv]\n"
            "# $8 = H2O volume mixing ratio [ppv]\n"
            "# $9 = O3 volume mixing ratio [ppv]\n"
            "# $10 = observed BT index [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;
      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;
    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_3d(met0, met0->t, met1, met1->t, t, press, lon,
                           lat, &temp, ci, cw, 1);
        intpol_met_time_3d(met0, met0->h2o, met1, met1->
h2o, t, press, lon,
                           lat, &h2o, ci, cw, 0);
        intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
                           lat, &o3, ci, cw, 0);

        /* 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]);
      }
    }

  /* 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];

  /* 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 (int 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 (int 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 (int 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 cache_t

  Cache data.
- struct met_t

  Meteorological data.

Functions

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

  Convert Cartesian coordinates to geolocation.
- int check_finite (const double x)

  Check if x is finite.
- double clim_hno3 (double t, double lat, double p)

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

  Climatology of OH number concentrations.
- 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_space_3d (met_t ∗met, float array[EX][EY][EP], double p, double lon, double lat, double ∗var, int ∗ci, double ∗cw, int init)

    Spatial interpolation of meteorological data.

- void intpol_met_space_2d (met_t ∗met, float array[EX][EY], double lon, double lat, double ∗var, int ∗ci, double ∗cw, int init)

    Spatial interpolation of meteorological data.

- void intpol_met_time_3d (met_t ∗met0, float array0[EX][EY][EP], met_t ∗met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double ∗var, int ∗ci, double ∗cw, int init)

    Temporal interpolation of meteorological data.

- void intpol_met_time_2d (met_t ∗met0, float array0[EX][EY], met_t ∗met1, float array1[EX][EY], double ts, double lon, double lat, double ∗var, int ∗ci, double ∗cw, int init)

    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_cloud (met_t ∗met)

    Calculate cloud properties.

- void read_met_extrapolate (met_t ∗met)

    Extrapolate meteorological data at lower boundary.

- void read_met_geopot (met_t ∗met)

    Calculate geopotential heights.

- int read_met_help_3d (int ncid, char ∗varname, char ∗varname2, met_t ∗met, float dest[EX][EY][EP], float scl)

    Read and convert 3D variable from meteorological data file.

- int read_met_help_2d (int ncid, char ∗varname, char ∗varname2, met_t ∗met, float dest[EX][EY], float scl)

    Read and convert 2D 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_surface (int ncid, met_t ∗met)

Read surface 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 spline (double ∗x, double ∗y, int n, double ∗x2, double ∗y2, int n2)

  Spline interpolation.

- double stddev (double ∗data, int n)

  Calculate standard deviation.

- 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 = 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 int check_finite (const double *x*)

Check if x is finite.

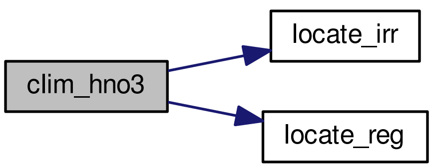
5.21.2.3 double clim_hno3 (double *t*, double *lat*, double *p*)

Climatology of HNO3 volume mixing ratios.

Definition at line 295 of file libtrac.c.

```
              {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Check pressure... */
  if (p < clim_hno3_ps[0])
    p = clim_hno3_ps[0];
  else if (p > clim_hno3_ps[9])
    p = clim_hno3_ps[9];

  /* Get indices... */
  int isec = locate_irr(clim_hno3_secs, 12, sec);
  int ilat = locate_reg(clim_hno3_lats, 18, lat);
  int ip = locate_irr(clim_hno3_ps, 10, p);

  /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
  double aux00 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec][ilat][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec][ilat + 1][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec + 1][ilat][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim_hno3_ps[ip],
                     clim_hno3_var[isec + 1][ilat + 1][ip],
                     clim_hno3_ps[ip + 1],
                     clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim_hno3_lats[ilat], aux00,
              clim_hno3_lats[ilat + 1], aux01, lat);
  aux11 = LIN(clim_hno3_lats[ilat], aux10,
              clim_hno3_lats[ilat + 1], aux11, lat);
  return LIN(clim_hno3_secs[isec], aux00,
             clim_hno3_secs[isec + 1], aux11, sec);
}
```

Here is the call graph for this function:



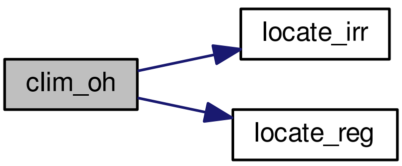
5.21.2.4 double clim_oh (double *t*, double *lat*, double *p*)

Climatology of OH number concentrations.

Definition at line 1322 of file libtrac.c.

```
           {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Check pressure... */
  if (p < clim_oh_ps[0])
    p = clim_oh_ps[0];
  else if (p > clim_oh_ps[33])
    p = clim_oh_ps[33];

  /* Get indices... */
  int isec = locate_irr(clim_oh_secs, 12, sec);
  int ilat = locate_reg(clim_oh_lats, 18, lat);
  int ip = locate_irr(clim_oh_ps, 34, p);

  /* Interpolate OH climatology (Pommrich et al., 2014)... */
  double aux00 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec][ilat][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec][ilat + 1][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec + 1][ilat][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim_oh_ps[ip],
                     clim_oh_var[isec + 1][ilat + 1][ip],
                     clim_oh_ps[ip + 1],
                     clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
  aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
  return 1e6 * LIN(clim_oh_secs[isec], aux00,
                   clim_oh_secs[isec + 1], aux11, sec);
}
```

Here is the call graph for this function:



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

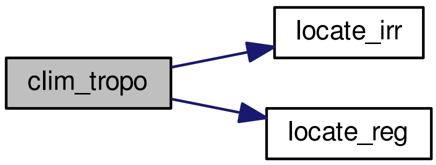
Climatology of tropopause pressure.

Definition at line 1497 of file libtrac.c.

```
             {

  /* Get seconds since begin of year... */
  double sec = FMOD(t, 365.25 * 86400.);
  while (sec < 0)
    sec += 365.25 * 86400.;

  /* Get indices... */
  int isec = locate_irr(clim_tropo_secs, 12, sec);
```

```
  int ilat = locate_reg(clim_tropo_lats, 73, lat);

  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
  double p0 = LIN(clim_tropo_lats[ilat],
                  clim_tropo_tps[isec][ilat],
                  clim_tropo_lats[ilat + 1],
                  clim_tropo_tps[isec][ilat + 1], lat);
  double p1 = LIN(clim_tropo_lats[ilat],
                  clim_tropo_tps[isec + 1][ilat],
                  clim_tropo_lats[ilat + 1],
                  clim_tropo_tps[isec + 1][ilat + 1], lat);
  return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
}
```

Here is the call graph for this function:



5.21.2.6 void day2doy (int *year,* int *mon,* int *day,* int ∗ *doy*)

Get day of year from date.

Definition at line 1524 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;
}
```

5.21.2.7 void doy2day (int *year,* int *doy,* int ∗ *mon,* int ∗ *day*)

Get date from day of year.

Definition at line 1542 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 };
  int i;

  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d0l[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0l[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1570 of file libtrac.c.

```
              {

  double 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);
}
```

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

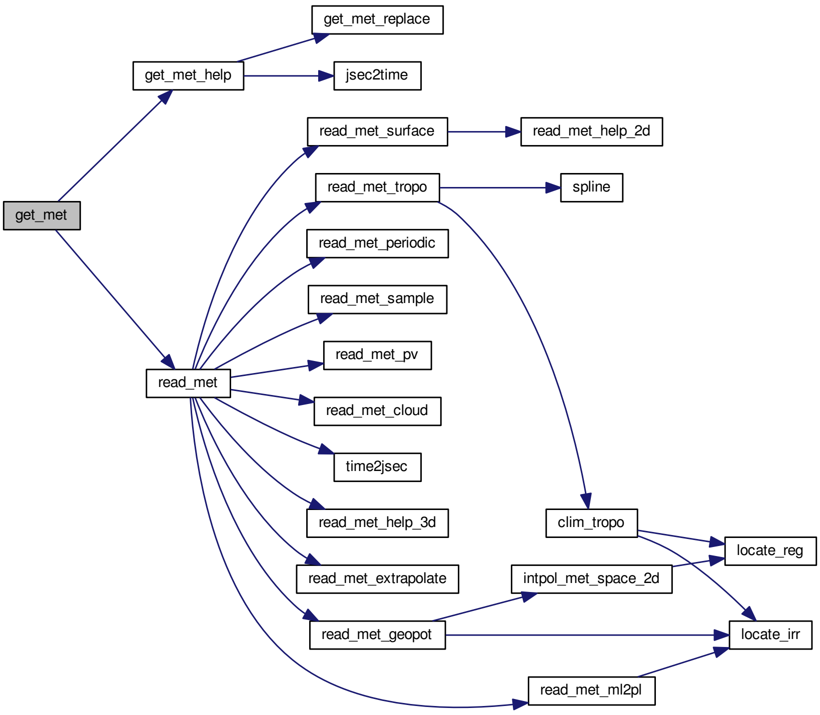
Get meteorological data for given timestep.

Definition at line 1584 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!");
#ifdef _OPENACC
    met_t *met0up = *met0;
    met_t *met1up = *met1;
#pragma acc update device(met0up[:1],met1up[:1])
#endif
  }

  /* 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!");
#ifdef _OPENACC
    met_t *met1up = *met1;
#pragma acc update device(met1up[:1])
#endif
  }

  /* 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!");
#ifdef _OPENACC
    met_t *met0up = *met0;
#pragma acc update device(met0up[:1])
#endif
  }

  /* 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:



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

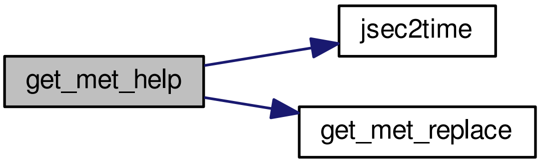
Get meteorological data for timestep.

Definition at line 1660 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:



5.21.2.11 void get_met_replace (char ∗ *orig,* char ∗ *search,* char ∗ *repl*)

Replace template strings in filename.

Definition at line 1696 of file libtrac.c.

```
             {

  char buffer[LEN], *ch;

  /* Iterate... */
  for (int 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);
  }
}
```

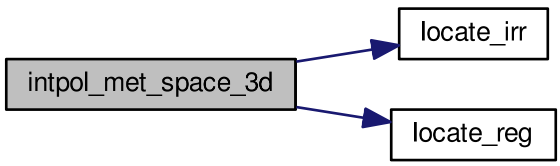
5.21.2.12 void intpol_met_space_3d (met_t ∗ *met,* float *array[EX][EY][EP],* double *p,* double *lon,* double *lat,* double ∗ *var,* int ∗ *ci,* double ∗ *cw,* int *init*)

Spatial interpolation of meteorological data.

Definition at line 1719 of file libtrac.c.

```
            {

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

  /* Get interpolation indices and weights... */
  if (init) {
    ci[0] = locate_irr(met->p, met->np, p);
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);
    cw[0] = (met->p[ci[0] + 1] - p)
      / (met->p[ci[0] + 1] - met->p[ci[0]]);
    cw[1] = (met->lon[ci[1] + 1] - lon)
      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
    cw[2] = (met->lat[ci[2] + 1] - lat)
      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
  }

  /* Interpolate vertically... */
  double aux00 =
    cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
    + array[ci[1]][ci[2]][ci[0] + 1];
  double aux01 =
    cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
             array[ci[1]][ci[2] + 1][ci[0] + 1])
    + array[ci[1]][ci[2] + 1][ci[0] + 1];
  double aux10 =
    cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
             array[ci[1] + 1][ci[2]][ci[0] + 1])
    + array[ci[1] + 1][ci[2]][ci[0] + 1];
  double aux11 =
    cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
    + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];

  /* Interpolate horizontally... */
  aux00 = cw[2] * (aux00 - aux01) + aux01;
  aux11 = cw[2] * (aux10 - aux11) + aux11;
  *var = cw[1] * (aux00 - aux11) + aux11;
}
```

Here is the call graph for this function:



5.21.2.13 void intpol_met_space_2d (met_t ∗ *met,* float *array[EX][EY],* double *lon,* double *lat,* double ∗ *var,* int ∗ *ci,* double ∗ *cw,* int *init*)

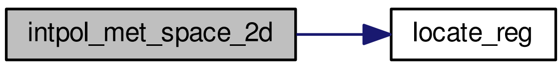
Spatial interpolation of meteorological data.

Definition at line 1773 of file libtrac.c.

```
            {

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

```

```
  /* Get interpolation indices and weights... */
  if (init) {
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);
    cw[1] = (met->lon[ci[1] + 1] - lon)
      / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
    cw[2] = (met->lat[ci[2] + 1] - lat)
      / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
  }

  /* Set variables... */
  double aux00 = array[ci[1]][ci[2]];
  double aux01 = array[ci[1]][ci[2] + 1];
  double aux10 = array[ci[1] + 1][ci[2]];
  double aux11 = array[ci[1] + 1][ci[2] + 1];

  /* Interpolate horizontally... */
  if (isfinite(aux00) && isfinite(aux01))
    aux00 = cw[2] * (aux00 - aux01) + aux01;
  else if (cw[2] < 0.5)
    aux00 = aux01;
  if (isfinite(aux10) && isfinite(aux11))
    aux11 = cw[2] * (aux10 - aux11) + aux11;
  else if (cw[2] > 0.5)
    aux11 = aux10;
  if (isfinite(aux00) && isfinite(aux11))
    *var = cw[1] * (aux00 - aux11) + aux11;
  else {
    if (cw[1] > 0.5)
      *var = aux00;
    else
      *var = aux11;
  }
}
```

Here is the call graph for this function:



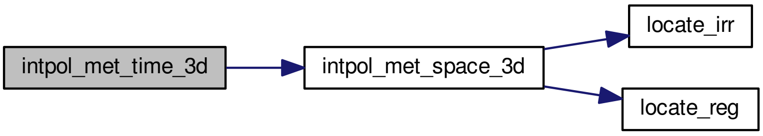
5.21.2.14 void intpol_met_time_3d (met_t * *met0*, float *array0[EX][EY][EP]*, met_t * *met1*, float *array1[EX][EY][EP]*, double *ts*, double *p*, double *lon*, double *lat*, double * *var*, int * *ci*, double * *cw*, int *init*)

Temporal interpolation of meteorological data.

Definition at line 1824 of file libtrac.c.

```
            {

  double var0, var1, wt;

  /* Spatial interpolation... */
  intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
  intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);

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

  /* Interpolate... */
  *var = wt * (var0 - var1) + var1;
}
```

Here is the call graph for this function:



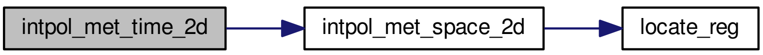
5.21.2.15 void intpol_met_time_2d (met_t * *met0*, float *array0[EX][EY]*, met_t * *met1*, float *array1[EX][EY]*, double *ts*, double *lon*, double *lat*, double * *var*, int * *ci*, double * *cw*, int *init*)

Temporal interpolation of meteorological data.

Definition at line 1853 of file libtrac.c.

```
            {

  double var0, var1, wt;

  /* Spatial interpolation... */
  intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
  intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);

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

  /* Interpolate... */
  *var = wt * (var0 - var1) + var1;
}
```

Here is the call graph for this function:



5.21.2.16 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 1881 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;

  time_t 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);
}
```

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

Find array index for irregular grid.

Definition at line 1914 of file libtrac.c.

```
             {

  int ilo = 0;
  int ihi = n - 1;
  int 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;
}
```

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

Find array index for regular grid.

Definition at line 1944 of file libtrac.c.

```
             {

  /* Calculate index... */
  int 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;
}
```

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

Read atmospheric data.

Definition at line 1963 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"))) {
      WARN("File not found!");
      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;
}
```

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

Read control parameters.

Definition at line 2143 of file libtrac.c.

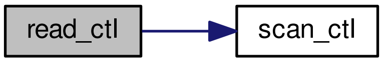
```
                {

  /* 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_lwc = -1;
  ctl->qnt_iwc = -1;
  ctl->qnt_pc = -1;
  ctl->qnt_hno3 = -1;
  ctl->qnt_oh = -1;
  ctl->qnt_rh = -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 (int 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], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
      ctl->qnt_o3 = iq;
      sprintf(ctl->qnt_unit[iq], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
      ctl->qnt_lwc = iq;
      sprintf(ctl->qnt_unit[iq], "kg/kg");
    } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
      ctl->qnt_iwc = iq;
      sprintf(ctl->qnt_unit[iq], "kg/kg");
    } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
      ctl->qnt_pc = iq;
      sprintf(ctl->qnt_unit[iq], "hPa");
    } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
      ctl->qnt_hno3 = iq;
      sprintf(ctl->qnt_unit[iq], "ppv");
    } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
      ctl->qnt_oh = iq;
      sprintf(ctl->qnt_unit[iq], "molec/cm^3");
    } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
      ctl->qnt_rh = iq;
      sprintf(ctl->qnt_unit[iq], "%%");
    } 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 (int 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, "3", NULL);
  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);

  /* Species parameters... */
  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
  if (strcmp(ctl->species, "SO2") == 0) {
    ctl->molmass = 64.066;
    ctl->oh_chem[0] = 3.3e-31;   /* (JPL Publication 15-10) */
    ctl->oh_chem[1] = 4.3;       /* (JPL Publication 15-10) */
    ctl->oh_chem[2] = 1.6e-12;   /* (JPL Publication 15-10) */
    ctl->oh_chem[3] = 0.0;       /* (JPL Publication 15-10) */
    ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
    ctl->wet_depo[1] = 0.62;    /* (FLEXPART v10.4) */
    ctl->wet_depo[2] = 1.3e-2;  /* (Sander, 2015) */
    ctl->wet_depo[3] = 2900.0;  /* (Sander, 2015) */
  } else {
    ctl->molmass =
      scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", 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);
    for (int ip = 0; ip < 4; ip++)
      ctl->oh_chem[ip] =
        scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
    for (int ip = 0; ip < 4; ip++)
      ctl->wet_depo[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO", ip, "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_stride =
    (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", 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:



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

Read meteorological data file.

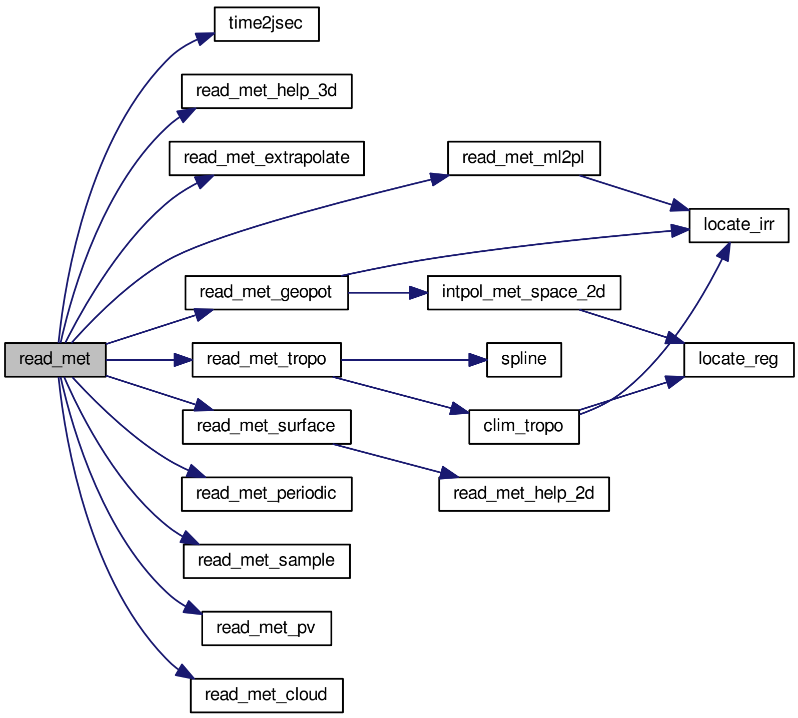
Definition at line 2455 of file libtrac.c.

```
                 {

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

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

  size_t np, nx, ny;

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

  /* 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);

  /* 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) {
      WARN("File not found!");
      return 0;
    }
  }

  /* 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");
    if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
```

```
      sprintf(levname, "plev");
      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... */
  if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
    ERRMSG("Cannot read temperature!");
  if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
    ERRMSG("Cannot read zonal wind!");
  if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
    ERRMSG("Cannot read meridional wind!");
  if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
    WARN("Cannot read vertical velocity");
  if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
    WARN("Cannot read specific humidity!");
  if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
    WARN("Cannot read ozone data!");
  if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
    WARN("Cannot read cloud liquid water content!");
  if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
    WARN("Cannot read cloud ice water content!");

  /* 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_3d(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);
    read_met_ml2pl(ctl, met, met->lwc);
    read_met_ml2pl(ctl, met, met->iwc);

    /* 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 data... */
  read_met_surface(ncid, met);

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

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

  /* Calculate geopotential heights... */
```

```
  read_met_geopot(met);

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

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

  /* Calculate cloud properties... */
  read_met_cloud(met);

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

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

Here is the call graph for this function:



5.21.2.22 void read_met_cloud (met_t * *met*)

Calculate cloud properties.

Definition at line 2622 of file libtrac.c.

```
                {

  int ix, iy, ip;

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

```

```
      /* Init... */
      met->pc[ix][iy] = GSL_NAN;
      met->cl[ix][iy] = 0;

      /* Loop over pressure levels... */
      for (ip = 0; ip < met->np - 1; ip++) {

        /* Check pressure... */
        if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
          continue;

        /* Get cloud top pressure ... */
        if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
          met->pc[ix][iy] = (float) met->p[ip + 1];

        /* Get cloud water... */
        met->cl[ix][iy] += (float)
          (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
                  + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
           * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
      }
    }
}
```

5.21.2.23 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 2658 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 (!isfinite(met->t[ix][iy][ip0])
            || !isfinite(met->u[ix][iy][ip0])
            || !isfinite(met->v[ix][iy][ip0])
            || !isfinite(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];
        met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
        met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
      }
    }
}
```

5.21.2.24 void read_met_geopot (met_t * *met*)

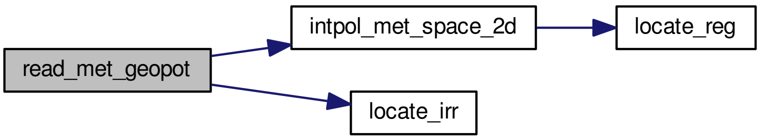
Calculate geopotential heights.

Definition at line 2692 of file libtrac.c.

```
                {

  const int dx = 6, dy = 4;

  static float help[EX][EY][EP];

  double logp[EP], ts, z0, cw[3];
```

```

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];

  /* Calculate log pressure... */
  for (ip = 0; ip < met->np; ip++)
    logp[ip] = log(met->p[ip]);

  /* 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;

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

      /* Get surface height... */
      intpol_met_space_2d(met, met->zs, met->lon[ix], met->
lat[iy], &z0, ci,
                          cw, 1);

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

      /* Get virtual temperature at the surface... */
      ts =
        LIN(met->p[ip0],
            TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
            met->p[ip0 + 1],
            TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[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 + TVIRT(met->t[ix][iy][ip0 + 1],
                                 met->h2o[ix][iy][ip0 + 1]))
                   * (log(met->ps[ix][iy]) - logp[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 *
                     (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
                      + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
                     * (logp[ip - 1] - logp[ip]));
    }

  /* Horizontal smoothing... */
#pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        n = 0;
        help[ix][iy][ip] = 0;
        for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - dy, 0);
               iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
            if (isfinite(met->z[ix3][iy2][ip])) {
              help[ix][iy][ip] += met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0)
          help[ix][iy][ip] /= (float) n;
        else
          help[ix][iy][ip] = GSL_NAN;
      }

  /* Copy 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++)
        met->z[ix][iy][ip] = help[ix][iy][ip];
}
```

Here is the call graph for this function:



5.21.2.25 int read_met_help_3d (int *ncid,* char ∗ *varname,* char ∗ *varname2,* met_t ∗ *met,* float *dest[EX][EY][EP],* float *scl*)

Read and convert 3D variable from meteorological data file.

Definition at line 2784 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 0;

  /* Allocate... */
  ALLOC(help, float, EX * EY * EP);

  /* 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);

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

5.21.2.26 int read_met_help_2d (int *ncid,* char ∗ *varname,* char ∗ *varname2,* met_t ∗ *met,* float *dest[EX][EY],* float *scl*)

Read and convert 2D variable from meteorological data file.

Definition at line 2828 of file libtrac.c.

```
             {

  float *help;

  int 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 0;

  /* Allocate... */
  ALLOC(help, float, EX * EY);

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

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

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

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

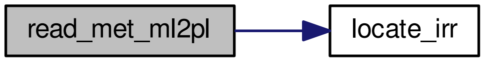
5.21.2.27 void read_met_ml2pl (ctl_t * ctl, met_t * met, float var[EX][EY][EP])

Convert meteorological data from model levels to pressure levels.

Definition at line 2871 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:



5.21.2.28 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 2910 of file libtrac.c.

```
                {

  /* 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)
  for (int iy = 0; iy < met->ny; iy++) {
    met->ps[met->nx - 1][iy] = met->ps[0][iy];
    met->zs[met->nx - 1][iy] = met->zs[0][iy];
    for (int ip = 0; ip < met->np; 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->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
      met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
      met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
    }
  }
}
```

5.21.2.29 void read_met_pv (met_t * *met*)

Calculate potential vorticity.

Definition at line 2945 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 = 0.5 * (met->lat[iy1] + met->lat[iy0]);
      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));
      }
    }
  }

  /* Fix for polar regions... */
#pragma omp parallel for default(shared) private(ix,ip)
  for (ix = 0; ix < met->nx; ix++)
    for (ip = 0; ip < met->np; ip++) {
      met->pv[ix][0][ip]
        = met->pv[ix][1][ip]
        = met->pv[ix][2][ip];
      met->pv[ix][met->ny - 1][ip]
        = met->pv[ix][met->ny - 2][ip]
        = met->pv[ix][met->ny - 3][ip];
    }
}
```

5.21.2.30 void read_met_sample (ctl_t ∗ *ctl*, met_t ∗ *met*)

Downsampling of meteorological data.

Definition at line 3045 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->zs[ix][iy] = 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->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        help->lwc[ix][iy][ip] = 0;
        help->iwc[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 - abs(ix - ix2) / ctl->met_sx)
                * (float) (1.0 - abs(iy - iy2) / ctl->met_sy)
                * (float) (1.0 - abs(ip - ip2) / ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->zs[ix][iy] += w * met->zs[ix3][iy2];
              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->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
              help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->zs[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
        help->lwc[ix][iy][ip] /= wsum;
        help->iwc[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->zs[met->nx][met->ny] = help->zs[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl->met_dp) {
        met->p[met->np] = help->p[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->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->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
        met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

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

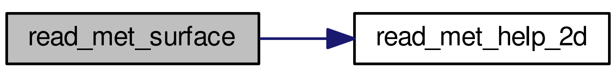
5.21.2.31 void read_met_surface (int *ncid*, met_t * *met*)

Read surface data.

Definition at line 3160 of file libtrac.c.

```
                {

  int ix, iy;

  /* Read surface pressure... */
  if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
    if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
      ERRMSG("Cannot not read surface pressure data!");
      for (ix = 0; ix < met->nx; ix++)
        for (iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) met->p[0];
    } else {
      for (iy = 0; iy < met->ny; iy++)
        for (ix = 0; ix < met->nx; ix++)
          met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
    }
  }

  /* Read geopotential height at the surface... */
  if (!read_met_help_2d
      (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
    if (!read_met_help_2d
        (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
      ERRMSG("Cannot read surface geopotential height!");
}
```

Here is the call graph for this function:



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

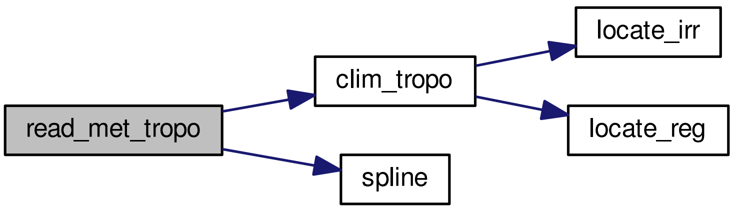
Calculate tropopause pressure.

Definition at line 3190 of file libtrac.c.

```
{

double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
  th2[200], z[EP], z2[200];

int found, ix, iy, iz, iz2;

/* Get altitude and pressure profiles... */
for (iz = 0; iz < met->np; iz++)
  z[iz] = Z(met->p[iz]);
for (iz = 0; iz <= 190; 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) {
#pragma omp parallel for default(shared) private(ix,iy)
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
}

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

  /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
  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];
      spline(z, t, met->np, z2, t2, 171);

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

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

  /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
  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];
      spline(z, t, met->np, z2, t2, 191);

      /* Find 1st tropopause... */
      met->pt[ix][iy] = GSL_NAN;
      for (iz = 0; iz <= 170; iz++) {
        found = 1;
        for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
          if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
              * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
            found = 0;
            break;
          }
        if (found) {
          if (iz > 0 && iz < 170)
            met->pt[ix][iy] = (float) p2[iz];
```

```
            break;
          }
        }

        /* Find 2nd tropopause... */
        if (ctl->met_tropo == 4) {
          met->pt[ix][iy] = GSL_NAN;
          for (; iz <= 170; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 170; iz++) {
            found = 1;
            for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
                found = 0;
                break;
              }
            if (found) {
              if (iz > 0 && iz < 170)
                met->pt[ix][iy] = (float) p2[iz];
              break;
            }
          }
        }
      }
  }

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

    /* Loop over grid points... */
#pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
    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];
        spline(z, pv, met->np, z2, pv2, 171);

        /* Interpolate potential temperature profile... */
        for (iz = 0; iz < met->np; iz++)
          th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
        spline(z, th, met->np, z2, th2, 171);

        /* Find dynamical tropopause 3.5 PVU + 380 K */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++)
          if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
            if (iz > 0 && iz < 170)
              met->pt[ix][iy] = (float) p2[iz];
            break;
          }
      }
  }

  else
    ERRMSG("Cannot calculate tropopause!");
}
```

Here is the call graph for this function:



5.21.2.33 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 3340 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);
}
```

5.21.2.34 void spline (double ∗ *x*, double ∗ *y*, int *n*, double ∗ *x2*, double ∗ *y2*, int *n2*)

Spline interpolation.

Definition at line 3412 of file libtrac.c.

```
          {

  gsl_interp_accel *acc;

  gsl_spline *s;

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);

  /* Interpolate temperature profile... */
  gsl_spline_init(s, x, y, (size_t) n);
  for (int i = 0; i < n2; i++)
    if (x2[i] <= x[0])
      y2[i] = y[0];
    else if (x2[i] >= x[n - 1])
      y2[i] = y[n - 1];
    else
      y2[i] = gsl_spline_eval(s, x2[i], acc);

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

5.21.2.35 double stddev (double ∗ *data*, int *n*)

Calculate standard deviation.

Definition at line 3445 of file libtrac.c.

```
         {

  if (n <= 0)
    return 0;

  double avg = 0, rms = 0;

  for (int i = 0; i < n; ++i)
    avg += data[i];
  avg /= n;

  for (int i = 0; i < n; ++i)
    rms += SQR(data[i] - avg);

  return sqrt(rms / (n - 1));
}
```

5.21.2.36 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 3466 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;
}
```

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

Measure wall-clock time.

Definition at line 3497 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;
  }
}
```

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

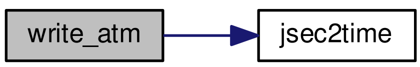
Write atmospheric data.

Definition at line 3533 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 += ctl->atm_stride) {

      /* 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:



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

Write CSI data.

Definition at line 3657 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);
}
```

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

Write ensemble data.

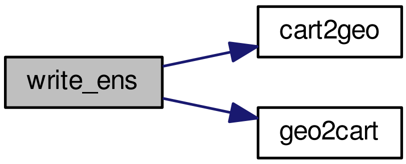
Definition at line 3827 of file libtrac.c.

```
                 {

  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:



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

Write gridded data.

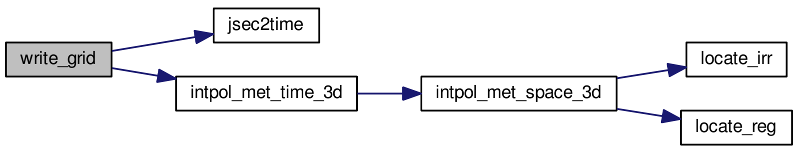
Definition at line 3958 of file libtrac.c.

```
            {

 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, cw[3];

 static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
   ci[3];

 /* 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 [ppv]\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_3d(met0, met0->t, met1, met1->t, t, press, lon,
                             lat, &temp, ci, cw, 1);

          /* 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 = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
            / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;

          /* 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:



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

Write profile data.

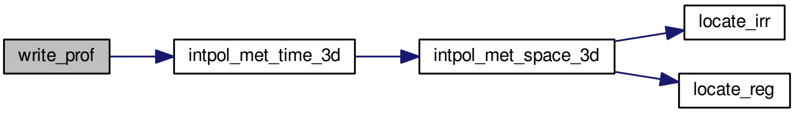
Definition at line 4112 of file libtrac.c.

```
             {

  static FILE *in, *out;

  static char line[LEN];

  static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
    t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
    o3, cw[3];

  static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];

  /* 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!");

    /* Check molar mass... */
    if (ctl->molmass <= 0)
      ERRMSG("Specify molar mass!");

    /* 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 [ppv]\n"
            "# $8 = H2O volume mixing ratio [ppv]\n"
            "# $9 = O3 volume mixing ratio [ppv]\n"
            "# $10 = observed BT index [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;
      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;
    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_3d(met0, met0->t, met1, met1->t, t, press, lon,
                             lat, &temp, ci, cw, 1);
          intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press, lon,
                             lat, &h2o, ci, cw, 0);
          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
                             lat, &o3, ci, cw, 0);

          /* 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]);
        }
      }

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

Here is the call graph for this function:



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

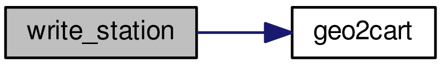
Write station data.

Definition at line 4295 of file libtrac.c.

```
             {

  static FILE *out;

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

  /* 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 (int 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 (int 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 (int 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
*/

/* ------------------------------------------------------------
   Includes...
   ------------------------------------------------------------ */

#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>

#ifdef MPI
#include "mpi.h"
#endif

#ifdef _OPENACC
#include "openacc.h"
```

```
#include "curand.h"
#endif

/* ------------------------------------------------------------
   Constants...
   ------------------------------------------------------------ */

#define G0 9.80665

#define H0 7.0

#define KB 1.3806504e-23

#define MA 28.9644

#define MH2O 18.01528

#define MO3 48.00

#define P0 1013.25

#define T0 273.15

#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 FMOD(x, y)                                      \
  ((x) - (int) ((x) / (y)) * (y))

#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 RH(p, t, h2o) (0.263 * 100. * (p) * MH2O / MA * (h2o)   \
                       / exp(17.67 * ((t) - T0) / ((t) - 29.65)))

#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 TVIRT(t, h2o) ((t) * (1.0 + 0.609133 * (h2o) * MH2O / MA))

#define WARN(msg) {                                                     \
    printf("\nWarning (%s, %s, l%d): %s\n\n",                           \
           __FILE__, __func__, __LINE__, msg);                          \
  }

#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 20

#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

#define TIMER_OHCHEM 12

#define TIMER_WETDEPO 13

#define TIMER_TOTAL 14

/* ------------------------------------------------------------
   NVIDIA Tools Extension (NVTX)...
   ------------------------------------------------------------ */

#ifdef USE_NVTX
#include "nvToolsExt.h"

#define NVTX_CPU 0xFFADD8E6

#define NVTX_GPU 0xFF00008B

#define NVTX_H2D 0xFFFFFF00

#define NVTX_D2H 0xFFFF8800

#define NVTX_READ 0xFFFFCCCB

#define NVTX_WRITE 0xFF8B0000

#define NVTX_MISC 0xFF808080

#define RANGE_PUSH(range_title, range_color) {          \
    nvtxEventAttributes_t eventAttrib = {0};            \
    eventAttrib.version = NVTX_VERSION;                 \
    eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;   \
    eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII;  \
    eventAttrib.colorType = NVTX_COLOR_ARGB;            \
    eventAttrib.color = range_color;                    \
    eventAttrib.message.ascii = range_title;            \
    nvtxRangePushEx(&eventAttrib);                      \
  }

#define RANGE_POP {                             \
    nvtxRangePop();                             \
  }
#else

/* Empty definitions of RANGE_PUSH and RANGE_POP... */
#define RANGE_PUSH(range_title, range_color) {}
#define RANGE_POP {}
#endif

/* ------------------------------------------------------------
   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_lwc;
int qnt_iwc;
int qnt_pc;
int qnt_hno3;
int qnt_oh;
int qnt_rh;
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;
```

```
char species[LEN];
double molmass;
double tdec_trop;
double tdec_strat;
double oh_chem[4];
double wet_depo[4];
double psc_h2o;
double psc_hno3;
char atm_basename[LEN];
char atm_gpfile[LEN];
double atm_dt_out;
int atm_filter;
int atm_stride;
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];

} atm_t;

typedef struct {

  float up[NP];

  float vp[NP];

  float wp[NP];

  double iso_var[NP];

  double iso_ps[NP];

  double iso_ts[NP];

  int iso_n;

  double tsig[EX][EY][EP];

  float usig[EX][EY][EP];

  float vsig[EX][EY][EP];

  float wsig[EX][EY][EP];

} cache_t;

typedef struct {

  double time;

  int nx;

  int ny;

  int np;

  double lon[EX];

  double lat[EY];

  double p[EP];

```

```
  float ps[EX][EY];

  float zs[EX][EY];

  float pt[EX][EY];

  float pc[EX][EY];

  float cl[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 lwc[EX][EY][EP];

  float iwc[EX][EY][EP];

  float pl[EX][EY][EP];

} met_t;

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

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

#ifdef _OPENACC
#pragma acc routine (check_finite)
#endif
int check_finite(
  const double x);

#ifdef _OPENACC
#pragma acc routine (clim_hno3)
#endif
double clim_hno3(
  double t,
  double lat,
  double p);

#ifdef _OPENACC
#pragma acc routine (clim_oh)
#endif
double clim_oh(
  double t,
  double lat,
  double p);

#ifdef _OPENACC
#pragma acc routine (clim_tropo)
#endif
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);

#ifdef _OPENACC
#pragma acc routine (intpol_met_space_3d)
#endif
void intpol_met_space_3d(
  met_t * met,
  float array[EX][EY][EP],
  double p,
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init);

#ifdef _OPENACC
#pragma acc routine (intpol_met_space_2d)
#endif
void intpol_met_space_2d(
  met_t * met,
  float array[EX][EY],
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init);

#ifdef _OPENACC
#pragma acc routine (intpol_met_time_3d)
#endif
void intpol_met_time_3d(
  met_t * met0,
  float array0[EX][EY][EP],
  met_t * met1,
  float array1[EX][EY][EP],
  double ts,
  double p,
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init);

#ifdef _OPENACC
#pragma acc routine (intpol_met_time_2d)
#endif
void intpol_met_time_2d(
  met_t * met0,
  float array0[EX][EY],
  met_t * met1,
  float array1[EX][EY],
  double ts,
  double lon,
  double lat,
  double *var,
  int *ci,
  double *cw,
  int init);

void jsec2time(
  double jsec,
  int *year,
  int *mon,
```

```
  int *day,
  int *hour,
  int *min,
  int *sec,
  double *remain);

#ifdef _OPENACC
#pragma acc routine (locate_irr)
#endif
int locate_irr(
  double *xx,
  int n,
  double x);

#ifdef _OPENACC
#pragma acc routine (locate_reg)
#endif
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_cloud(
  met_t * met);

void read_met_extrapolate(
  met_t * met);

void read_met_geopot(
  met_t * met);

int read_met_help_3d(
  int ncid,
  char *varname,
  char *varname2,
  met_t * met,
  float dest[EX][EY][EP],
  float scl);

int read_met_help_2d(
  int ncid,
  char *varname,
  char *varname2,
  met_t * met,
  float dest[EX][EY],
  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_surface(
  int ncid,
  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 spline(
  double *x,
  double *y,
  int n,
  double *x2,
  double *y2,
  int n2);

#ifdef _OPENACC
#pragma acc routine (stddev)
#endif
double stddev(
  double *data,
  int n);

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 map from meteorological data.

Functions

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

5.23.1 Detailed Description

Extract 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[ ]*)

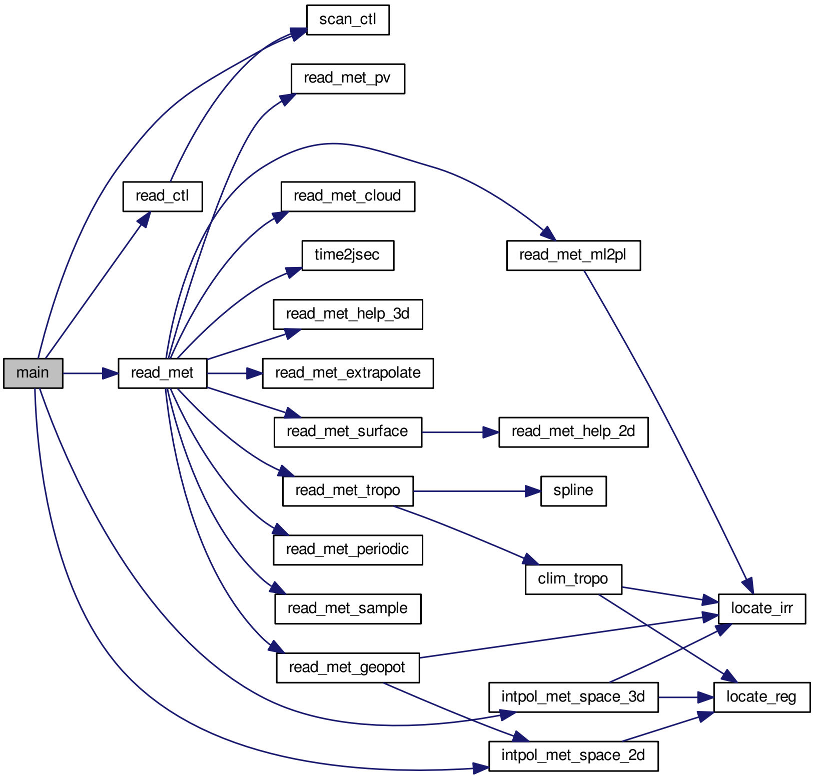
Definition at line 41 of file met_map.c.

```
                 {

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[NX][NY], p0, ps, psm[NX][NY], pt, ptm[NX][NY], t,
    tm[NX][NY], u, um[NX][NY], v, vm[NX][NY], w, wm[NX][NY], h2o,
    h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
    lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
    pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY],
    pc, pcm[NX][NY], cl, clm[NX][NY], lon, lon0, lon1, lons[NX],
    dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];

  static int i, ix, iy, np[NX][NY], nx, ny, ci[3];

  /* 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, "10", 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) > NX)
        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) > NY)
        ERRMSG("Too many latitudes!");
    }

    /* Average... */
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {

        /* Interpolate meteo data... */
        intpol_met_space_3d(met, met->z, p0, lons[ix], lats[iy], &z, ci, cw,
                            1);
        intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy], &t, ci, cw,
                            0);
        intpol_met_space_3d(met, met->u, p0, lons[ix], lats[iy], &u, ci, cw,
                            0);
        intpol_met_space_3d(met, met->v, p0, lons[ix], lats[iy], &v, ci, cw,
                            0);
        intpol_met_space_3d(met, met->w, p0, lons[ix], lats[iy], &w, ci, cw,
                            0);
        intpol_met_space_3d(met, met->pv, p0, lons[ix], lats[iy], &pv, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->h2o, p0, lons[ix], lats[iy], &h2o, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->o3, p0, lons[ix], lats[iy], &o3, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->lwc, p0, lons[ix], lats[iy], &lwc, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->iwc, p0, lons[ix], lats[iy], &iwc, ci,
                            cw, 0);
        intpol_met_space_2d(met, met->ps, lons[ix], lats[iy], &ps, ci, cw, 0);
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy], &pt, ci, cw, 0);
        intpol_met_space_2d(met, met->pc, lons[ix], lats[iy], &pc, ci, cw, 0);
        intpol_met_space_2d(met, met->cl, lons[ix], lats[iy], &cl, ci, cw, 0);

        /* Interpolate tropopause data... */
        intpol_met_space_3d(met, met->z, pt, lons[ix], lats[iy], &zt, ci, cw,
                            1);
        intpol_met_space_3d(met, met->t, pt, lons[ix], lats[iy], &tt, ci, cw,
                            0);
        intpol_met_space_3d(met, met->h2o, pt, lons[ix], lats[iy], &h2ot, ci,
                            cw, 0);

        /* Averaging... */
        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;
        lwcm[ix][iy] += lwc;
        iwcm[ix][iy] += iwc;
        psm[ix][iy] += ps;
        ptm[ix][iy] += pt;
        pcm[ix][iy] += pc;
        clm[ix][iy] += cl;
        ztm[ix][iy] += zt;
        ttm[ix][iy] += tt;
        h2otm[ix][iy] += h2ot;
        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 [ppv]\n");
fprintf(out,
        "# $11 = O3 volume mixing ratio [ppv]\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"
        "# $18 = tropopause water vapor [ppv]\n"
        "# $19 = cloud liquid water content [kg/kg]\n"
        "# $20 = cloud ice water content [kg/kg]\n");
fprintf(out,
        "# $21 = total column cloud water [kg/m^2]\n"
        "# $22 = cloud top pressure [hPa]\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 %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],
            h2otm[ix][iy] / np[ix][iy], lwcm[ix][iy] / np[ix][iy],
            iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
            pcm[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"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define NX 1441

#define NY 721

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
```

```
  int argc,
  char *argv[]) {

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[NX][NY], p0, ps, psm[NX][NY], pt, ptm[NX][NY], t,
    tm[NX][NY], u, um[NX][NY], v, vm[NX][NY], w, wm[NX][NY], h2o,
    h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
    lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
    pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY],
    pc, pcm[NX][NY], cl, clm[NX][NY], lon, lon0, lon1, lons[NX],
    dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];

  static int i, ix, iy, np[NX][NY], nx, ny, ci[3];

  /* 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, "10", 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) > NX)
        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) > NY)
        ERRMSG("Too many latitudes!");
    }

    /* Average... */
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {

        /* Interpolate meteo data... */
        intpol_met_space_3d(met, met->z, p0, lons[ix], lats[iy], &z, ci, cw,
                            1);
        intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy], &t, ci, cw,
                            0);
        intpol_met_space_3d(met, met->u, p0, lons[ix], lats[iy], &u, ci, cw,
                            0);
        intpol_met_space_3d(met, met->v, p0, lons[ix], lats[iy], &v, ci, cw,
                            0);
        intpol_met_space_3d(met, met->w, p0, lons[ix], lats[iy], &w, ci, cw,
                            0);
        intpol_met_space_3d(met, met->pv, p0, lons[ix], lats[iy], &pv, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->h2o, p0, lons[ix], lats[iy], &h2o, ci,
                            cw, 0);
```

```
        intpol_met_space_3d(met, met->o3, p0, lons[ix], lats[iy], &o3, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->lwc, p0, lons[ix], lats[iy], &lwc, ci,
                            cw, 0);
        intpol_met_space_3d(met, met->iwc, p0, lons[ix], lats[iy], &iwc, ci,
                            cw, 0);
        intpol_met_space_2d(met, met->ps, lons[ix], lats[iy], &ps, ci, cw, 0);
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy], &pt, ci, cw, 0);
        intpol_met_space_2d(met, met->pc, lons[ix], lats[iy], &pc, ci, cw, 0);
        intpol_met_space_2d(met, met->cl, lons[ix], lats[iy], &cl, ci, cw, 0);

        /* Interpolate tropopause data... */
        intpol_met_space_3d(met, met->z, pt, lons[ix], lats[iy], &zt, ci, cw,
                            1);
        intpol_met_space_3d(met, met->t, pt, lons[ix], lats[iy], &tt, ci, cw,
                            0);
        intpol_met_space_3d(met, met->h2o, pt, lons[ix], lats[iy], &h2ot, ci,
                            cw, 0);

        /* Averaging... */
        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;
        lwcm[ix][iy] += lwc;
        iwcm[ix][iy] += iwc;
        psm[ix][iy] += ps;
        ptm[ix][iy] += pt;
        pcm[ix][iy] += pc;
        clm[ix][iy] += cl;
        ztm[ix][iy] += zt;
        ttm[ix][iy] += tt;
        h2otm[ix][iy] += h2ot;
        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 [ppv]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\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 %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],
```

```
            h2otm[ix][iy] / np[ix][iy], lwcm[ix][iy] / np[ix][iy],
            iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
            pcm[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[ ]*)

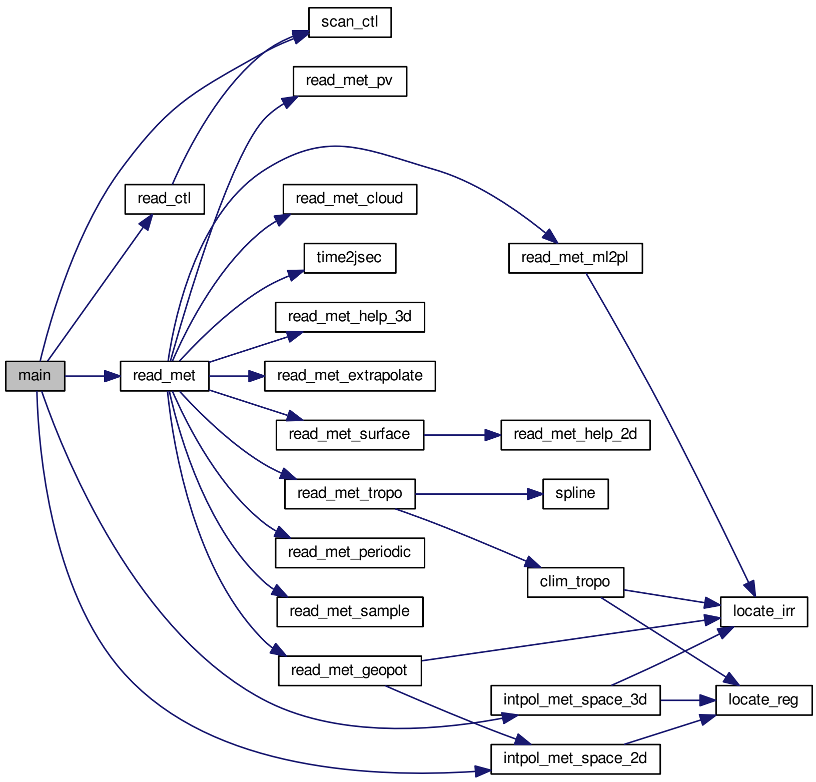
Definition at line 38 of file met_prof.c.

```
                  {

  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], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
    iwc, iwcm[NZ], ps, psm[NZ], pt, ptm[NZ], pc, pcm[NZ], cl, clm[NZ],
    tt, ttm[NZ], zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ], cw[3];

  static int i, iz, np[NZ], npt[NZ], nz, ci[3];

  /* 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, "-999", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
  dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
```

```
  lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", 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 vertical grid... */
    if (z0 < 0)
      z0 = Z(met->p[0]);
    if (z1 < 0)
      z1 = Z(met->p[met->np - 1]);
    nz = 0;
    if (dz < 0) {
      for (iz = 0; iz < met->np; iz++)
        if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
          plev[nz] = met->p[iz];
          if ((++nz) > NZ)
            ERRMSG("Too many pressure levels!");
        }
    } else
      for (z = z0; z <= z1; z += dz) {
        plev[nz] = P(z);
        if ((++nz) > NZ)
          ERRMSG("Too many pressure levels!");
      }

    /* 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]);

    /* Average... */
    for (iz = 0; iz < nz; iz++)
      for (lon = lon0; lon <= lon1; lon += dlon)
        for (lat = lat0; lat <= lat1; lat += dlat) {

          /* Interpolate meteo data... */
          intpol_met_space_3d(met, met->z, plev[iz], lon, lat, &z, ci, cw, 1);
          intpol_met_space_3d(met, met->t, plev[iz], lon, lat, &t, ci, cw, 0);
          intpol_met_space_3d(met, met->u, plev[iz], lon, lat, &u, ci, cw, 0);
          intpol_met_space_3d(met, met->v, plev[iz], lon, lat, &v, ci, cw, 0);
          intpol_met_space_3d(met, met->w, plev[iz], lon, lat, &w, ci, cw, 0);
          intpol_met_space_3d(met, met->pv, plev[iz], lon, lat, &pv, ci, cw,
                              0);
          intpol_met_space_3d(met, met->h2o, plev[iz], lon, lat, &h2o, ci, cw,
                              0);
          intpol_met_space_3d(met, met->o3, plev[iz], lon, lat, &o3, ci, cw,
                              0);
          intpol_met_space_3d(met, met->lwc, plev[iz], lon, lat, &lwc, ci, cw,
                              0);
          intpol_met_space_3d(met, met->iwc, plev[iz], lon, lat, &iwc, ci, cw,
                              0);
          intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
          intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
          intpol_met_space_2d(met, met->pc, lon, lat, &pc, ci, cw, 0);
          intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0);

          /* Interpolate tropopause data... */
          intpol_met_space_3d(met, met->z, pt, lon, lat, &zt, ci, cw, 1);
          intpol_met_space_3d(met, met->t, pt, lon, lat, &tt, ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, pt, lon, lat, &h2ot, ci, cw, 0);

          /* Averaging... */
          if (gsl_finite(t) && gsl_finite(u)
              && gsl_finite(v) && gsl_finite(w)) {
            timem[iz] += met->time;
            lonm[iz] += lon;
            latm[iz] += lat;
            zm[iz] += z;
            tm[iz] += t;
            um[iz] += u;
            vm[iz] += v;
            wm[iz] += w;
            pvm[iz] += pv;
            h2om[iz] += h2o;
            o3m[iz] += o3;
            psm[iz] += ps;
            pcm[iz] += pc;
            clm[iz] += cl;
            lwcm[iz] += lwc;
            iwcm[iz] += iwc;
```

```
          if (gsl_finite(pt)) {
            ptm[iz] += pt;
            ztm[iz] += zt;
            ttm[iz] += tt;
            h2otm[iz] += h2ot;
            npt[iz]++;
          }
          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"
          "# $10 = H2O volume mixing ratio [ppv]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\n\n");

  /* Write data... */
  for (iz = 0; iz < nz; iz++)
    fprintf(out,
            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
            timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
            latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
            vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
            o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
            psm[iz] / np[iz], ptm[iz] / npt[iz], ztm[iz] / npt[iz],
            ttm[iz] / npt[iz], h2otm[iz] / npt[iz], lwcm[iz] / np[iz],
            iwcm[iz] / np[iz], clm[iz] / np[iz], pcm[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...
   ------------------------------------------------------------ */

#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], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
    iwc, iwcm[NZ], ps, psm[NZ], pt, ptm[NZ], pc, pcm[NZ], cl, clm[NZ],
    tt, ttm[NZ], zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ], cw[3];

  static int i, iz, np[NZ], npt[NZ], nz, ci[3];

  /* 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, "-999", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
  dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
  lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", 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 vertical grid... */
    if (z0 < 0)
      z0 = Z(met->p[0]);
    if (z1 < 0)
      z1 = Z(met->p[met->np - 1]);
    nz = 0;
    if (dz < 0) {
      for (iz = 0; iz < met->np; iz++)
        if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
          plev[nz] = met->p[iz];
          if ((++nz) > NZ)
            ERRMSG("Too many pressure levels!");
        }
    } else
      for (z = z0; z <= z1; z += dz) {
        plev[nz] = P(z);
        if ((++nz) > NZ)
          ERRMSG("Too many pressure levels!");
      }

    /* 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]);

    /* Average... */
    for (iz = 0; iz < nz; iz++)
      for (lon = lon0; lon <= lon1; lon += dlon)
        for (lat = lat0; lat <= lat1; lat += dlat) {

          /* Interpolate meteo data... */
          intpol_met_space_3d(met, met->z, plev[iz], lon, lat, &z, ci, cw, 1);
          intpol_met_space_3d(met, met->t, plev[iz], lon, lat, &t, ci, cw, 0);
          intpol_met_space_3d(met, met->u, plev[iz], lon, lat, &u, ci, cw, 0);
          intpol_met_space_3d(met, met->v, plev[iz], lon, lat, &v, ci, cw, 0);
          intpol_met_space_3d(met, met->w, plev[iz], lon, lat, &w, ci, cw, 0);
          intpol_met_space_3d(met, met->pv, plev[iz], lon, lat, &pv, ci, cw,
                              0);
          intpol_met_space_3d(met, met->h2o, plev[iz], lon, lat, &h2o, ci, cw,
                              0);
          intpol_met_space_3d(met, met->o3, plev[iz], lon, lat, &o3, ci, cw,
                              0);
          intpol_met_space_3d(met, met->lwc, plev[iz], lon, lat, &lwc, ci, cw,
                              0);
          intpol_met_space_3d(met, met->iwc, plev[iz], lon, lat, &iwc, ci, cw,
```

```
                              0);
          intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
          intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
          intpol_met_space_2d(met, met->pc, lon, lat, &pc, ci, cw, 0);
          intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0);

          /* Interpolate tropopause data... */
          intpol_met_space_3d(met, met->z, pt, lon, lat, &zt, ci, cw, 1);
          intpol_met_space_3d(met, met->t, pt, lon, lat, &tt, ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, pt, lon, lat, &h2ot, ci, cw, 0);

          /* Averaging... */
          if (gsl_finite(t) && gsl_finite(u)
              && gsl_finite(v) && gsl_finite(w)) {
            timem[iz] += met->time;
            lonm[iz] += lon;
            latm[iz] += lat;
            zm[iz] += z;
            tm[iz] += t;
            um[iz] += u;
            vm[iz] += v;
            wm[iz] += w;
            pvm[iz] += pv;
            h2om[iz] += h2o;
            o3m[iz] += o3;
            psm[iz] += ps;
            pcm[iz] += pc;
            clm[iz] += cl;
            lwcm[iz] += lwc;
            iwcm[iz] += iwc;
            if (gsl_finite(pt)) {
              ptm[iz] += pt;
              ztm[iz] += zt;
              ttm[iz] += tt;
              h2otm[iz] += h2ot;
              npt[iz]++;
            }
            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"
          "# $10 = H2O volume mixing ratio [ppv]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\n\n");

  /* Write data... */
  for (iz = 0; iz < nz; iz++)
    fprintf(out,
            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
            timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
            latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
            vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
            o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
            psm[iz] / np[iz], ptm[iz] / npt[iz], ztm[iz] / npt[iz],
            ttm[iz] / npt[iz], h2otm[iz] / npt[iz], lwcm[iz] / np[iz],
            iwcm[iz] / np[iz], clm[iz] / np[iz], pcm[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[ ]*)

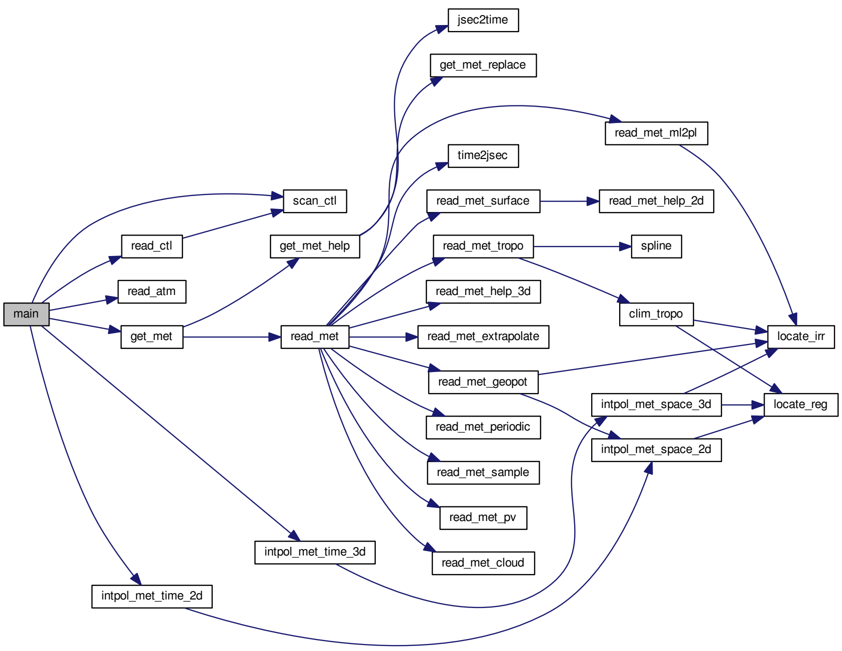
Definition at line 31 of file met_sample.c.

```
                {

  ctl_t ctl;

  atm_t *atm;

  met_t *met0, *met1;

  FILE *out;

  double h2o, h2ot, o3, lwc, iwc, p0, p1, pref, ps, pt, pc, cl, pv, t, tt, u,
    v, w, z, zm, zref, zt, cw[3];

  int geopot, ip, it, ci[3];

  /* 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"
          "# $10 = H2O volume mixing ratio [ppv]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\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_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
                           atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
        if (zref > zm || !gsl_finite(zm))
          p0 = pref;
        else
          p1 = pref;
      }
      pref = 0.5 * (p0 + p1);
    }

    /* Interpolate meteo data... */
    intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
    intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
    intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
    intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
    intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
    intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &pv, ci, cw, 0);
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &h2o, ci, cw, 0);
    intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &o3, ci, cw, 0);
    intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &lwc, ci, cw, 0);
    intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &iwc, ci, cw, 0);
    intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
```

```
    intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
time[ip],
                       atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
    intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
                       atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
    intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
                       atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);

    /* Interpolate tropopause data... */
    intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip], pt,
                       atm->lon[ip], atm->lat[ip], &zt, ci, cw, 1);
    intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip], pt,
                       atm->lon[ip], atm->lat[ip], &tt, ci, cw, 0);
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip], pt,
                       atm->lon[ip], atm->lat[ip], &h2ot, ci, cw, 0);

    /* Write data... */
    fprintf(out,
            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\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, h2ot, lwc,
            iwc, cl, pc);
  }

  /* 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, h2ot, o3, lwc, iwc, p0, p1, pref, ps, pt, pc, cl, pv, t, tt, u,
    v, w, z, zm, zref, zt, cw[3];

  int geopot, ip, it, ci[3];

  /* 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"
          "# $10 = H2O volume mixing ratio [ppv]\n");
  fprintf(out,
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
```

```
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\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_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
                           atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
        if (zref > zm || !gsl_finite(zm))
          p0 = pref;
        else
          p1 = pref;
      }
      pref = 0.5 * (p0 + p1);
    }

    /* Interpolate meteo data... */
    intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
    intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
    intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
    intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
    intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
    intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &pv, ci, cw, 0);
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &h2o, ci, cw, 0);
    intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &o3, ci, cw, 0);
    intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &lwc, ci, cw, 0);
    intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->time[ip], pref,
                       atm->lon[ip], atm->lat[ip], &iwc, ci, cw, 0);
    intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
    intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
    intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
    intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);

    /* Interpolate tropopause data... */
    intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pt,
                       atm->lon[ip], atm->lat[ip], &zt, ci, cw, 1);
    intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip], pt,
                       atm->lon[ip], atm->lat[ip], &tt, ci, cw, 0);
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->time[ip], pt,
                       atm->lon[ip], atm->lat[ip], &h2ot, ci, cw, 0);

    /* Write data... */
    fprintf(out,
            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\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, h2ot, lwc,
            iwc, cl, pc);
  }

  /* 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 41 of file met_zm.c.

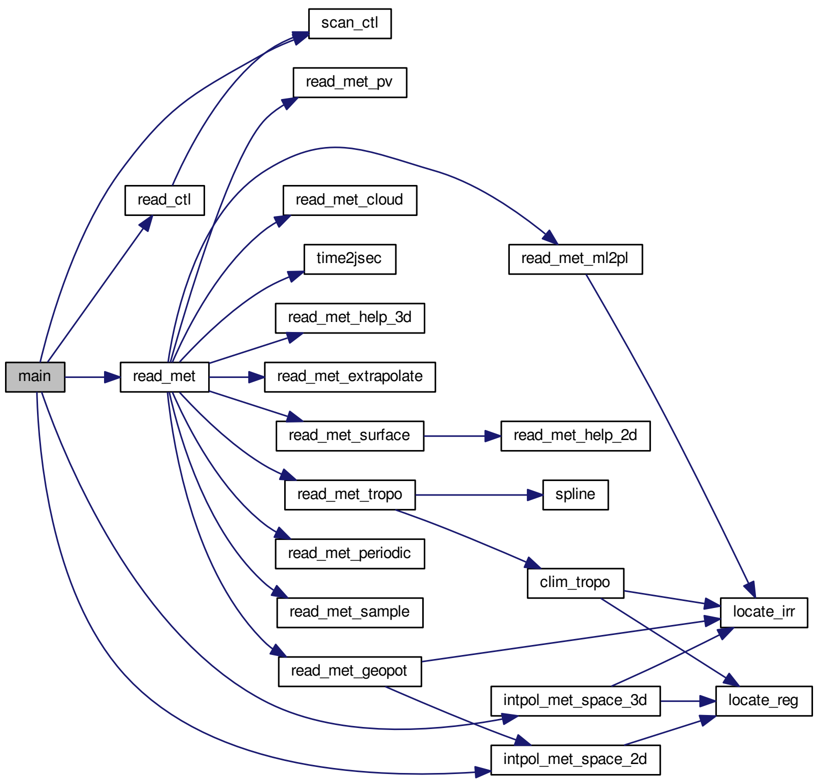
```
                {

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[NZ][NY], psm[NZ][NY], ptm[NZ][NY], pcm[NZ][NY],
    clm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY],
    wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY], pvm[NZ][NY], o3m[NZ][NY],
    lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
    ps, pt, pc, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc, lat, lat0, lat1,
    dlat, lats[NY], cw[3];

  static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz, ci[3];

  /* 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);
  z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
  dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
```

```
  lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "ZM_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 vertical grid... */
    if (z0 < 0)
      z0 = Z(met->p[0]);
    if (z1 < 0)
      z1 = Z(met->p[met->np - 1]);
    nz = 0;
    if (dz < 0) {
      for (iz = 0; iz < met->np; iz++)
        if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
          plev[nz] = met->p[iz];
          if ((++nz) > NZ)
            ERRMSG("Too many pressure levels!");
        }
    } else
      for (z = z0; z <= z1; z += dz) {
        plev[nz] = P(z);
        if ((++nz) > NZ)
          ERRMSG("Too many pressure levels!");
      }

    /* Set horizontal grid... */
    if (dlat <= 0)
      dlat = fabs(met->lat[1] - met->lat[0]);
    ny = 0;
    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) > NY)
        ERRMSG("Too many latitudes!");
    }

    /* Average... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < ny; iy++)
        for (iz = 0; iz < nz; iz++) {

          /* Interpolate meteo data... */
          intpol_met_space_3d(met, met->z, plev[iz], met->
      lon[ix],
                              met->lat[iy], &z, ci, cw, 1);
          intpol_met_space_3d(met, met->t, plev[iz], met->
      lon[ix],
                              met->lat[iy], &t, ci, cw, 0);
          intpol_met_space_3d(met, met->u, plev[iz], met->
      lon[ix],
                              met->lat[iy], &u, ci, cw, 0);
          intpol_met_space_3d(met, met->v, plev[iz], met->
      lon[ix],
                              met->lat[iy], &v, ci, cw, 0);
          intpol_met_space_3d(met, met->w, plev[iz], met->
      lon[ix],
                              met->lat[iy], &w, ci, cw, 0);
          intpol_met_space_3d(met, met->pv, plev[iz], met->
      lon[ix],
                              met->lat[iy], &pv, ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, plev[iz], met->
      lon[ix],
                              met->lat[iy], &h2o, ci, cw, 0);
          intpol_met_space_3d(met, met->o3, plev[iz], met->
      lon[ix],
                              met->lat[iy], &o3, ci, cw, 0);
          intpol_met_space_3d(met, met->lwc, plev[iz], met->
      lon[ix],
                              met->lat[iy], &lwc, ci, cw, 0);
          intpol_met_space_3d(met, met->iwc, plev[iz], met->
      lon[ix],
                              met->lat[iy], &iwc, ci, cw, 0);
          intpol_met_space_2d(met, met->ps, met->lon[ix], met->
      lat[iy], &ps,
                              ci, cw, 0);
          intpol_met_space_2d(met, met->pt, met->lon[ix], met->
      lat[iy], &pt,
                              ci, cw, 0);
```

```
        intpol_met_space_2d(met, met->pc, met->lon[ix], met->
    lat[iy], &pc,
                            ci, cw, 0);
        intpol_met_space_2d(met, met->cl, met->lon[ix], met->
    lat[iy], &cl,
                            ci, cw, 0);

        /* Interpolate tropopause data... */
        intpol_met_space_3d(met, met->z, pt, met->lon[ix], met->
    lat[iy],
                            &zt, ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt, met->lon[ix], met->
    lat[iy],
                            &tt, ci, cw, 0);
        intpol_met_space_3d(met, met->h2o, pt, met->lon[ix], met->
    lat[iy],
                            &h2ot, ci, cw, 0);

        /* Averaging... */
        timem[iz][iy] += met->time;
        zm[iz][iy] += z;
        tm[iz][iy] += t;
        um[iz][iy] += u;
        vm[iz][iy] += v;
        wm[iz][iy] += w;
        pvm[iz][iy] += pv;
        h2om[iz][iy] += h2o;
        o3m[iz][iy] += o3;
        lwcm[iz][iy] += lwc;
        iwcm[iz][iy] += iwc;
        psm[iz][iy] += ps;
        pcm[iz][iy] += pc;
        clm[iz][iy] += cl;
        if (gsl_finite(pt)) {
          ptm[iz][iy] += pt;
          ztm[iz][iy] += zt;
          ttm[iz][iy] += tt;
          h2otm[iz][iy] += h2ot;
          npt[iz][iy]++;
        }
        np[iz][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");
  fprintf(out,
          "# $10 = H2O volume mixing ratio [ppv]\n"
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\n");

  /* Write data... */
  for (iz = 0; iz < nz; iz++) {
    fprintf(out, "\n");
    for (iy = 0; iy < ny; iy++)
      fprintf(out,
              "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
              plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
              vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
              h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
              zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
              psm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
```

```
                ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
                h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
                iwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
                pcm[iz][iy] / np[iz][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"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define NZ 1000

#define NY 721

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

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

  ctl_t ctl;

  met_t *met;

  FILE *out;

  static double timem[NZ][NY], psm[NZ][NY], ptm[NZ][NY], pcm[NZ][NY],
    clm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY],
    wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY], pvm[NZ][NY], o3m[NZ][NY],
    lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
    ps, pt, pc, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc, lat, lat0, lat1,
    dlat, lats[NY], cw[3];

  static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz, ci[3];

  /* 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);
  z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
  dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "ZM_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 vertical grid... */
    if (z0 < 0)
      z0 = Z(met->p[0]);
    if (z1 < 0)
      z1 = Z(met->p[met->np - 1]);
    nz = 0;
    if (dz < 0) {
      for (iz = 0; iz < met->np; iz++)
        if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
          plev[nz] = met->p[iz];
          if ((++nz) > NZ)
            ERRMSG("Too many pressure levels!");
        }
    } else
      for (z = z0; z <= z1; z += dz) {
        plev[nz] = P(z);
        if ((++nz) > NZ)
          ERRMSG("Too many pressure levels!");
      }

    /* Set horizontal grid... */
    if (dlat <= 0)
      dlat = fabs(met->lat[1] - met->lat[0]);
    ny = 0;
    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) > NY)
        ERRMSG("Too many latitudes!");
    }

    /* Average... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < ny; iy++)
        for (iz = 0; iz < nz; iz++) {

          /* Interpolate meteo data... */
          intpol_met_space_3d(met, met->z, plev[iz], met->lon[ix],
                              met->lat[iy], &z, ci, cw, 1);
          intpol_met_space_3d(met, met->t, plev[iz], met->lon[ix],
                              met->lat[iy], &t, ci, cw, 0);
          intpol_met_space_3d(met, met->u, plev[iz], met->lon[ix],
                              met->lat[iy], &u, ci, cw, 0);
          intpol_met_space_3d(met, met->v, plev[iz], met->lon[ix],
                              met->lat[iy], &v, ci, cw, 0);
          intpol_met_space_3d(met, met->w, plev[iz], met->lon[ix],
                              met->lat[iy], &w, ci, cw, 0);
          intpol_met_space_3d(met, met->pv, plev[iz], met->lon[ix],
                              met->lat[iy], &pv, ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, plev[iz], met->lon[ix],
                              met->lat[iy], &h2o, ci, cw, 0);
          intpol_met_space_3d(met, met->o3, plev[iz], met->lon[ix],
                              met->lat[iy], &o3, ci, cw, 0);
          intpol_met_space_3d(met, met->lwc, plev[iz], met->lon[ix],
                              met->lat[iy], &lwc, ci, cw, 0);
          intpol_met_space_3d(met, met->iwc, plev[iz], met->lon[ix],
                              met->lat[iy], &iwc, ci, cw, 0);
          intpol_met_space_2d(met, met->ps, met->lon[ix], met->lat[iy], &ps,
                              ci, cw, 0);
          intpol_met_space_2d(met, met->pt, met->lon[ix], met->lat[iy], &pt,
                              ci, cw, 0);
          intpol_met_space_2d(met, met->pc, met->lon[ix], met->lat[iy], &pc,
                              ci, cw, 0);
          intpol_met_space_2d(met, met->cl, met->lon[ix], met->lat[iy], &cl,
                              ci, cw, 0);

          /* Interpolate tropopause data... */
          intpol_met_space_3d(met, met->z, pt, met->lon[ix], met->lat[iy],
                              &zt, ci, cw, 1);
          intpol_met_space_3d(met, met->t, pt, met->lon[ix], met->lat[iy],
                              &tt, ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, pt, met->lon[ix], met->lat[iy],
                              &h2ot, ci, cw, 0);

          /* Averaging... */
          timem[iz][iy] += met->time;
          zm[iz][iy] += z;
          tm[iz][iy] += t;
          um[iz][iy] += u;
          vm[iz][iy] += v;
          wm[iz][iy] += w;
          pvm[iz][iy] += pv;
          h2om[iz][iy] += h2o;
          o3m[iz][iy] += o3;
          lwcm[iz][iy] += lwc;
          iwcm[iz][iy] += iwc;
          psm[iz][iy] += ps;
          pcm[iz][iy] += pc;
          clm[iz][iy] += cl;
          if (gsl_finite(pt)) {
            ptm[iz][iy] += pt;
            ztm[iz][iy] += zt;
            ttm[iz][iy] += tt;
            h2otm[iz][iy] += h2ot;
            npt[iz][iy]++;
          }
```

```
        np[iz][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");
  fprintf(out,
          "# $10 = H2O volume mixing ratio [ppv]\n"
          "# $11 = O3 volume mixing ratio [ppv]\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"
          "# $18 = tropopause water vapor [ppv]\n"
          "# $19 = cloud liquid water content [kg/kg]\n"
          "# $20 = cloud ice water content [kg/kg]\n");
  fprintf(out,
          "# $21 = total column cloud water [kg/m^2]\n"
          "# $22 = cloud top pressure [hPa]\n");

  /* Write data... */
  for (iz = 0; iz < nz; iz++) {
    fprintf(out, "\n");
    for (iy = 0; iy < ny; iy++)
      fprintf(out,
              "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
              plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
              vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
              h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
              zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
              psm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
              ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
              h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
              iwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
              pcm[iz][iy] / np[iz][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.

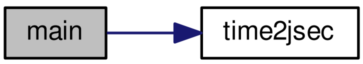
5.31.2 Function Documentation

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

Definition at line 27 of file time2jsec.c.

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

Here is the call graph for this function:



5.32 time2jsec.c

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

```
  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, double *dt)

  Calculate advection of air parcels.
- void module_decay (ctl_t *ctl, atm_t *atm, double *dt)

  Calculate exponential decay of particle mass.
- void module_diffusion_init (void)

  Initialize random number generator...
- void module_diffusion_meso (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, cache_t *cache, double *dt, double *rs)

  Calculate mesoscale diffusion.
- void module_diffusion_rng (double *rs, size_t n)

  Generate random numbers.
- void module_diffusion_turb (ctl_t *ctl, atm_t *atm, double *dt, double *rs)

  Calculate turbulent diffusion.
- void module_isosurf_init (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, cache_t *cache)

  Initialize isosurface module.
- void module_isosurf (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, cache_t *cache)

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

  Interpolate meteorological data for air parcel positions.
- void module_position (met_t *met0, met_t *met1, atm_t *atm, double *dt)

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

  Calculate sedimentation of air parcels.
- void module_oh_chem (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double *dt)

  Calculate OH chemistry.
- void module_wet_deposition (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double *dt)

  Calculate wet deposition.
- 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[ ])

Variables

- curandGenerator_t rng

5.33.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.33.2 Function Documentation

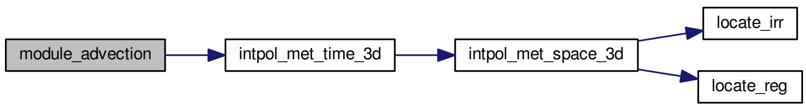
5.33.2.1 void module_advection (met_t * met0, met_t * met1, atm_t * atm, double * dt)

Calculate advection of air parcels.

Definition at line 507 of file trac.c.

```
                 {

#ifdef _OPENACC
#pragma acc data present(met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      int ci[3] = { 0 };

      double dtm = 0.0, v[3] = { 0.0 }, xm[3] = {
      0.0};
      double cw[3] = { 0.0 };

      /* Interpolate meteorological data... */
      intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[0], ci,
                         cw, 1);
      intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[1], ci,
                         cw, 0);
      intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[2], ci,
                         cw, 0);

      /* Get position of the mid point... */
      dtm = atm->time[ip] + 0.5 * dt[ip];
      xm[0] =
        atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
      xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
      xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];

      /* Interpolate meteorological data for mid point... */
      intpol_met_time_3d(met0, met0->u, met1, met1->u, dtm, xm[2], xm[0],
                         xm[1], &v[0], ci, cw, 1);
      intpol_met_time_3d(met0, met0->v, met1, met1->v, dtm, xm[2], xm[0],
                         xm[1], &v[1], ci, cw, 0);
      intpol_met_time_3d(met0, met0->w, met1, met1->w, dtm, xm[2], xm[0],
                         xm[1], &v[2], ci, cw, 0);

      /* Save new position... */
      atm->time[ip] += dt[ip];
      atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
      atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
      atm->p[ip] += dt[ip] * v[2];
    }
}
```

Here is the call graph for this function:



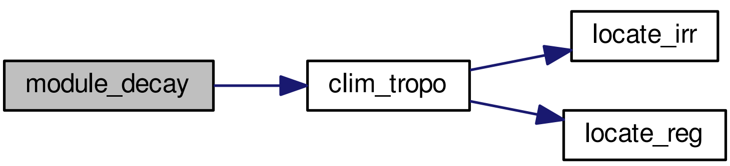
5.33.2.2 void module_decay (ctl_t * *ctl*, atm_t * *atm*, double * *dt*)

Calculate exponential decay of particle mass.

Definition at line 564 of file trac.c.

```
                {

  /* Check quantity flags... */
  if (ctl->qnt_m < 0)
    ERRMSG("Module needs quantity mass!");

#ifdef _OPENACC
#pragma acc data present(ctl,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double p0, p1, pt, tdec, 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 lifetime... */
      tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;

      /* Calculate exponential decay... */
      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
    }
}
```

Here is the call graph for this function:



5.33.2.3 void module_diffusion_init (void)

Initialize random number generator...

Definition at line 607 of file trac.c.

```
          {

   /* Initialize random number generator... */
#ifdef _OPENACC

   if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
       != CURAND_STATUS_SUCCESS)
     ERRMSG("Cannot create random number generator!");
   if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
       != CURAND_STATUS_SUCCESS)
     ERRMSG("Cannot set stream for random number generator!");

#else

   gsl_rng_env_setup();
   if (omp_get_max_threads() > NTHREADS)
     ERRMSG("Too many threads!");
   for (int 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);
   }

#endif
}
```

5.33.2.4 void module_diffusion_meso (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, cache_t * *cache*, double * *dt*, double * *rs*)

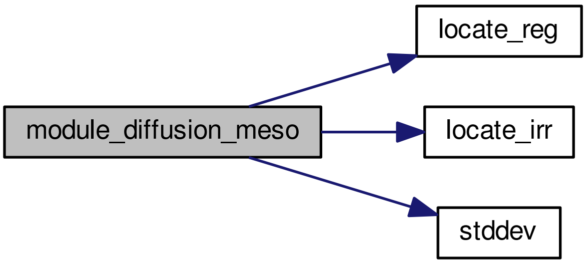
Calculate mesoscale diffusion.

Definition at line 635 of file trac.c.

```
                {

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
   for (int ip = 0; ip < atm->np; ip++)
     if (dt[ip] != 0) {

       double u[16], v[16], w[16];

       /* Get indices... */
       int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
       int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
       int iz = locate_irr(met0->p, met0->np, atm->p[ip]);

       /* Caching of wind standard deviations... */
       if (cache->tsig[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... */
      cache->usig[ix][iy][iz] = (float) stddev(u, 16);
      cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
      cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
      cache->tsig[ix][iy][iz] = met0->time;
    }

    /* Set temporal correlations for mesoscale fluctuations... */
    double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
    double r2 = sqrt(1 - r * r);

    /* Calculate horizontal mesoscale wind fluctuations... */
    if (ctl->turb_mesox > 0) {
      cache->up[ip] = (float)
        (r * cache->up[ip]
         + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
      atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);

      cache->vp[ip] = (float)
        (r * cache->vp[ip]
         + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
      atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
    }

    /* Calculate vertical mesoscale wind fluctuations... */
    if (ctl->turb_mesoz > 0) {
      cache->wp[ip] = (float)
        (r * cache->wp[ip]
         + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
      atm->p[ip] += cache->wp[ip] * dt[ip];
    }
  }
}
```

Here is the call graph for this function:



5.33.2.5 void module_diffusion_rng (double * *rs*, size_t *n*)

Generate random numbers.

Definition at line 755 of file trac.c.

```
                  {

#ifdef _OPENACC

#pragma acc host_data use_device(rs)
  {
    if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
        != CURAND_STATUS_SUCCESS)
      ERRMSG("Cannot create random numbers!");
  }

#else

#pragma omp parallel for default(shared)
  for (size_t i = 0; i < n; ++i)
    rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);

#endif

}
```

5.33.2.6 void module_diffusion_turb (ctl_t * *ctl*, atm_t * *atm*, double * *dt*, double * *rs*)

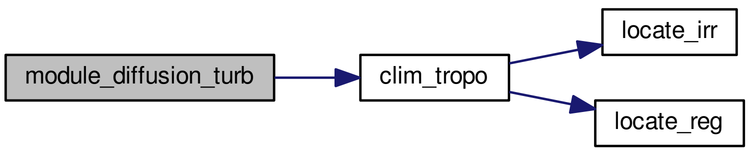
Calculate turbulent diffusion.

Definition at line 780 of file trac.c.

```
                    {

#ifdef _OPENACC
#pragma acc data present(ctl,atm,dt,rs)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double w;

```

```
      /* Get tropopause pressure... */
      double pt = clim_tropo(atm->time[ip], atm->lat[ip]);

      /* Get weighting factor... */
      double p1 = pt * 0.866877899;
      double 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... */
      double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
      double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;

      /* Horizontal turbulent diffusion... */
      if (dx > 0) {
        double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
        atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
        atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
      }

      /* Vertical turbulent diffusion... */
      if (dz > 0) {
        double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
        atm->p[ip]
          += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
      }
    }
}
```

Here is the call graph for this function:



5.33.2.7 void module_isosurf_init (ctl_t * *ctl,* met_t * *met0,* met_t * *met1,* atm_t * *atm,* cache_t * *cache*)

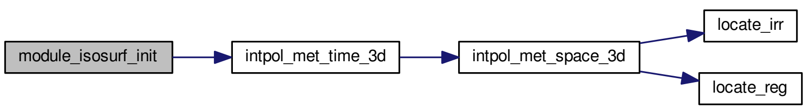
Initialize isosurface module.

Definition at line 832 of file trac.c.

```
                   {

  FILE *in;

  char line[LEN];

  double t, cw[3];

  int ci[3];

  /* Save pressure... */
  if (ctl->isosurf == 1)
    for (int ip = 0; ip < atm->np; ip++)
      cache->iso_var[ip] = atm->p[ip];

  /* Save density... */
```

```
  else if (ctl->isosurf == 2)
    for (int ip = 0; ip < atm->np; ip++) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                         1);
      cache->iso_var[ip] = atm->p[ip] / t;
    }

  /* Save potential temperature... */
  else if (ctl->isosurf == 3)
    for (int ip = 0; ip < atm->np; ip++) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                         1);
      cache->iso_var[ip] = THETA(atm->p[ip], 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", &(cache->iso_ts[cache->iso_n]),
                 &(cache->iso_ps[cache->iso_n])) == 2)
        if ((++cache->iso_n) > NP)
          ERRMSG("Too many data points!");

    /* Check number of points... */
    if (cache->iso_n < 1)
      ERRMSG("Could not read any data!");

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

Here is the call graph for this function:



5.33.2.8 void module_isosurf (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, cache_t * *cache*)

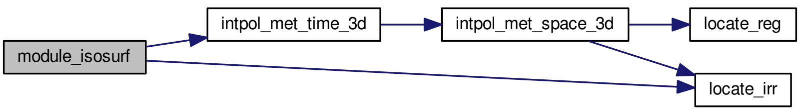
Force air parcels to stay on isosurface.

Definition at line 898 of file trac.c.

```
                        {

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,cache)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++) {

    double t, cw[3];
```

```

    int ci[3];

    /* Restore pressure... */
    if (ctl->isosurf == 1)
      atm->p[ip] = cache->iso_var[ip];

    /* Restore density... */
    else if (ctl->isosurf == 2) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                         1);
      atm->p[ip] = cache->iso_var[ip] * t;
    }

    /* Restore potential temperature... */
    else if (ctl->isosurf == 3) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                         1);
      atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
    }

    /* Interpolate pressure... */
    else if (ctl->isosurf == 4) {
      if (atm->time[ip] <= cache->iso_ts[0])
        atm->p[ip] = cache->iso_ps[0];
      else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
        atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
      else {
        int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
        atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
                         cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
                         atm->time[ip]);
      }
    }
  }
}
```

Here is the call graph for this function:



5.33.2.9 void module_meteo (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*)

Interpolate meteorological data for air parcel positions.

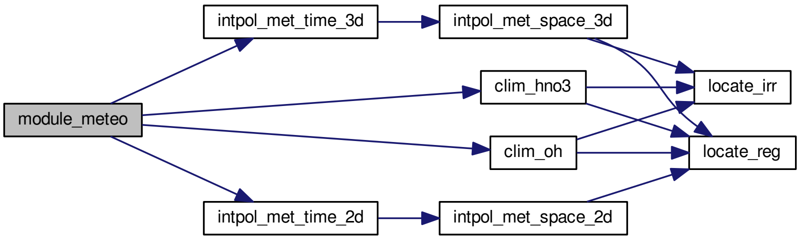
Definition at line 955 of file trac.c.

```
                  {

  /* Check quantity flags... */
  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!");

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++) {
```

```

  double ps, pt, pc, pv, t, u, v, w, h2o, o3, lwc, iwc, z, cw[3];

  int ci[3];

  /* Interpolate meteorological data... */
  intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
  intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
  intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
  intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
  intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
  intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &pv, ci, cw,
                     0);
  intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &h2o, ci, cw,
                     0);
  intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &o3, ci, cw,
                     0);
  intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
                     0);
  intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->time[ip],
                     atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
                     0);
  intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->time[ip],
                     atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
  intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->time[ip],
                     atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
  intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->time[ip],
                     atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);

  /* 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;

      /* Set cloud liquid water content... */
      if (ctl->qnt_lwc >= 0)
        atm->q[ctl->qnt_lwc][ip] = lwc;

      /* Set cloud ice water content... */
      if (ctl->qnt_iwc >= 0)
        atm->q[ctl->qnt_iwc][ip] = iwc;

      /* Set cloud top pressure... */
      if (ctl->qnt_pc >= 0)
        atm->q[ctl->qnt_pc][ip] = pc;

      /* Set nitric acid vmr... */
      if (ctl->qnt_hno3 >= 0)
        atm->q[ctl->qnt_hno3][ip] =
          clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]);

      /* Set hydroxyl number concentration... */
      if (ctl->qnt_oh >= 0)
        atm->q[ctl->qnt_oh][ip] =
          clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]);

      /* 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 relative humidty... */
      if (ctl->qnt_rh >= 0)
        atm->q[ctl->qnt_rh][ip] = RH(atm->p[ip], t, h2o);

      /* 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) {
        double p_hno3;
        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;
        double p_h2o =
          (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
        double a = 0.009179 - 0.00088 * log10(p_h2o);
        double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
        double c = -11397.0 / a;
        double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
        double 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)
        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:



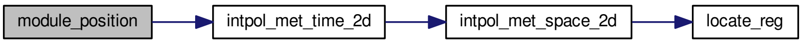
5.33.2.10 void module_position (met_t * *met0*, met_t * *met1*, atm_t * *atm*, double * *dt*)

Check position of air parcels.

Definition at line 1130 of file trac.c.

```
                    {

#ifdef _OPENACC
#pragma acc data present(met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double ps, cw[3];

      int ci[3];

      /* 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;

      /* Check pressure... */
      if (atm->p[ip] < met0->p[met0->np - 1])
        atm->p[ip] = met0->p[met0->np - 1];
      else if (atm->p[ip] > 300.) {
        intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->time[ip],
                           atm->lon[ip], atm->lat[ip], &ps, ci, cw, 1);
        if (atm->p[ip] > ps)
          atm->p[ip] = ps;
      }
    }
}
```

Here is the call graph for this function:



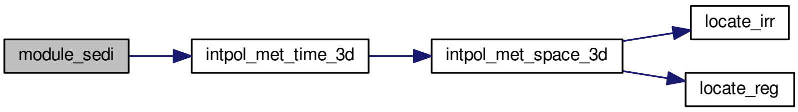
5.33.2.11 void module_sedi (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double * *dt*)

Calculate sedimentation of air parcels.

Definition at line 1185 of file trac.c.

```
               {

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p, cw[3];

      int ci[3];

      /* 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_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
                         1);

      /* 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 * 4.8096e-26));

      /* 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 * (1.249 + 0.42 * exp(-0.87 / 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[ip] / 1000., atm->p[ip]);
    }
}
```

Here is the call graph for this function:



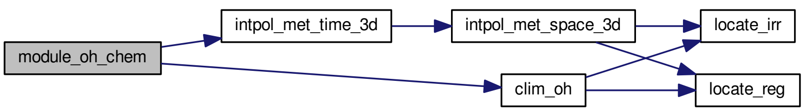
5.33.2.12 void module_oh_chem (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double * *dt*)

Calculate OH chemistry.

Definition at line 1243 of file trac.c.

```
            {

  /* Check quantity flags... */
  if (ctl->qnt_m < 0)
    ERRMSG("Module needs quantity mass!");

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double c, k, k0, ki, M, T, cw[3];

      int ci[3];

      /* Get temperature... */
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
                         1);

      /* Calculate molecular density... */
      M = 7.243e21 * (atm->p[ip] / P0) / T;

      /* Calculate rate coefficient for X + OH + M -> XOH + M
         (JPL Publication 15-10) ... */
      k0 = ctl->oh_chem[0] *
        (ctl->oh_chem[1] > 0 ? pow(T / 300., -ctl->oh_chem[1]) : 1.);
      ki = ctl->oh_chem[2] *
        (ctl->oh_chem[3] > 0 ? pow(T / 300., -ctl->oh_chem[3]) : 1.);
      c = log10(k0 * M / ki);
      k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));

      /* Calculate exponential decay... */
      atm->q[ctl->qnt_m][ip] *=
        exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->
p[ip]));
    }
}
```

Here is the call graph for this function:



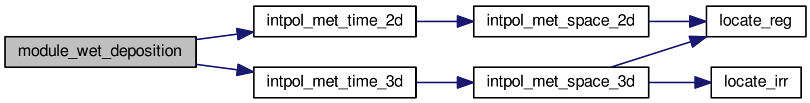
5.33.2.13 void module_wet_deposition (ctl_t * *ctl,* met_t * *met0,* met_t * *met1,* atm_t * *atm,* double * *dt*)

Calculate wet deposition.

Definition at line 1292 of file trac.c.

```
                     {

  /* Check quantity flags... */
  if (ctl->qnt_m < 0)
    ERRMSG("Module needs quantity mass!");

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double H, Is, Si, T, cl, lambda, iwc, lwc, pc, cw[3];

      int inside, ci[3];

      /* Check whether particle is below cloud top... */
      intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->time[ip],
                         atm->lon[ip], atm->lat[ip], &pc, ci, cw, 1);
      if (!isfinite(pc) || atm->p[ip] <= pc)
        continue;

      /* Check whether particle is inside or below cloud... */
      intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
                         1);
      intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
                         0);
      inside = (iwc > 0 || lwc > 0);

      /* Estimate precipitation rate (Pisso et al., 2019)... */
      intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->time[ip],
                         atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
      Is = pow(2. * cl, 1. / 0.36);
      if (Is < 0.01)
        continue;

      /* Calculate in-cloud scavenging for gases... */
      if (inside) {

        /* Get temperature... */
        intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                           atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
                           0);

        /* Get Henry's constant (Sander, 2015)... */
        H = ctl->wet_depo[2] * 101.325
          * exp(ctl->wet_depo[3] * (1. / T - 1. / 298.15));

        /* Get scavenging coefficient (Hertel et al., 1995)... */
        Si = 1. / ((1. - cl) / (H * RI / P0 * T) + cl);
        lambda = 6.2 * Si * Is / 3.6e6;
      }

      /* Calculate below-cloud scavenging for gases (Pisso et al., 2019)... */
      else
        lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);

      /* Calculate exponential decay... */
      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
    }
}
```

Here is the call graph for this function:



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

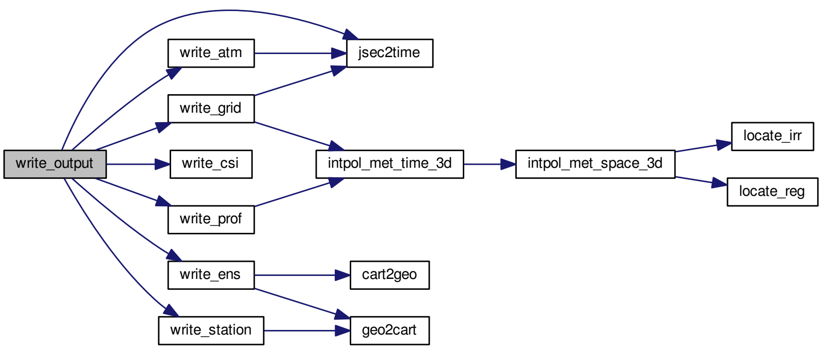
Write simulation output.

Definition at line 1366 of file trac.c.

```
            {

  char filename[2 * LEN];

  double r;

  int year, mon, day, hour, min, sec, updated = 0;

  /* 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) {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write atm data", NVTX_WRITE);
    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);
    RANGE_POP;
  }

  /* Write gridded data... */
  if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write grid data", NVTX_WRITE);
    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);
    RANGE_POP;
  }

  /* Write CSI data... */
  if (ctl->csi_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write CSI data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
    write_csi(filename, ctl, atm, t);
    RANGE_POP;
```

```
  }

  /* Write ensemble data... */
  if (ctl->ens_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write ensemble data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
    write_ens(filename, ctl, atm, t);
    RANGE_POP;
  }

  /* Write profile data... */
  if (ctl->prof_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write profile data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write_prof(filename, ctl, met0, met1, atm, t);
    RANGE_POP;
  }

  /* Write station data... */
  if (ctl->stat_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write station data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
    write_station(filename, ctl, atm, t);
    RANGE_POP;
  }
}
```

Here is the call graph for this function:



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

Definition at line 147 of file trac.c.

```
                {

ctl_t ctl;

atm_t *atm;

cache_t *cache;

met_t *met0, *met1;

FILE *dirlist;

char dirname[LEN], filename[2 * LEN];

double *dt, *rs, t;

int num_devices = 0, ntask = -1, rank = 0, size = 1;

/* Initialize MPI... */
#ifdef MPI
RANGE_PUSH("Initialize MPI", NVTX_CPU);
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
RANGE_POP;
#endif

/* Initialize GPUs... */
#ifdef _OPENACC
RANGE_PUSH("Initialize GPUs", NVTX_GPU);
acc_device_t device_type = acc_get_device_type();
num_devices = acc_get_num_devices(acc_device_nvidia);
int device_num = rank % num_devices;
acc_set_device_num(device_num, acc_device_nvidia);
acc_init(device_type);
RANGE_POP;
#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... */
  RANGE_PUSH("Allocate", NVTX_CPU);
  ALLOC(atm, atm_t, 1);
  ALLOC(cache, cache_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);
  ALLOC(dt, double,
        NP);
  ALLOC(rs, double,
        3 * NP);
  RANGE_POP;

  /* Create data region on GPUs... */
#ifdef _OPENACC
  RANGE_PUSH("Create data region", NVTX_GPU);
#pragma acc enter data create(atm[:1],cache[:1],ctl,met0[:1],met1[:1],dt[:NP],rs[:3*NP])
  RANGE_POP;
#endif

  /* Read control parameters... */
  RANGE_PUSH("Read ctl", NVTX_READ);
  sprintf(filename, "%s/%s", dirname, argv[2]);
  read_ctl(filename, argc, argv, &ctl);
  RANGE_POP;

  /* Read atmospheric data... */
```

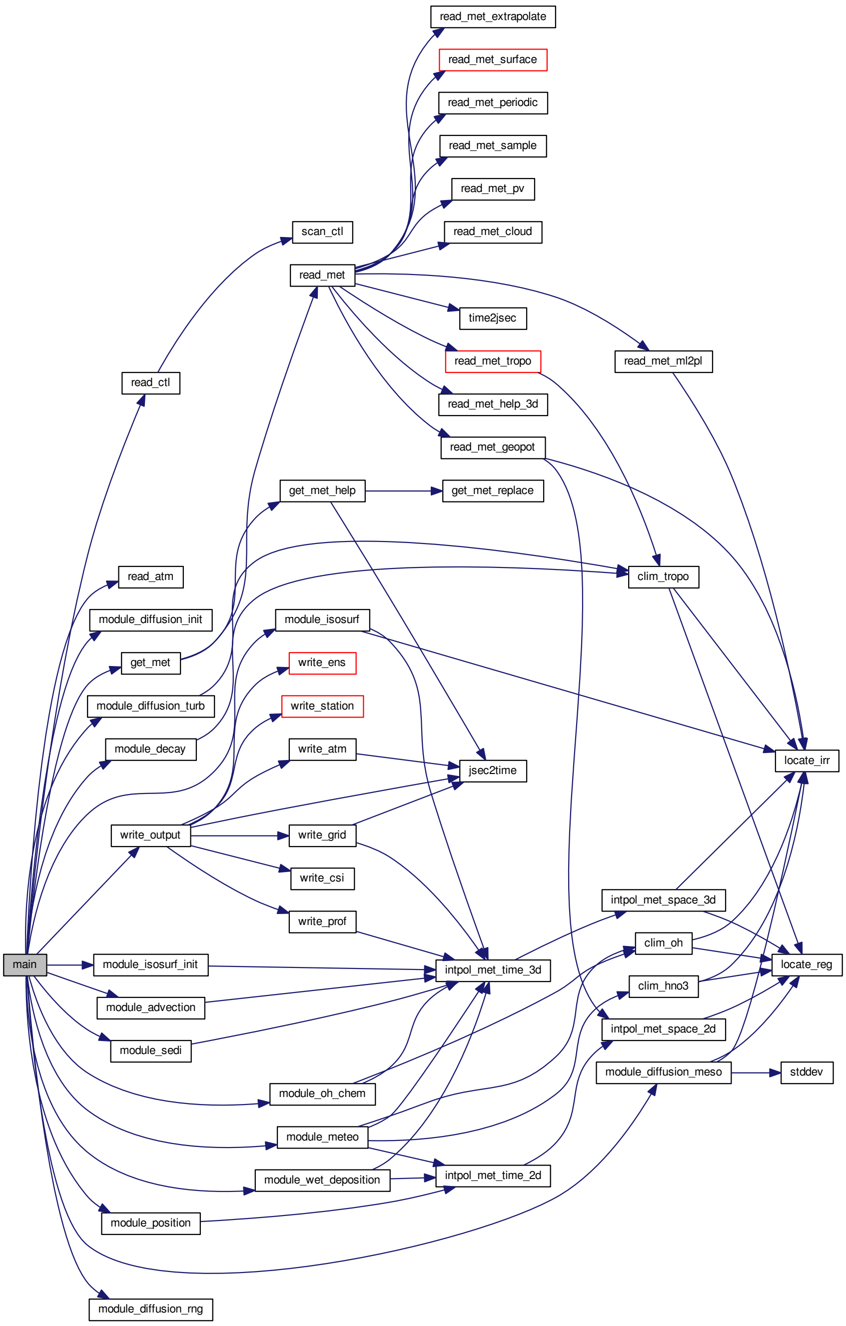
```
    RANGE_PUSH("Read atm", NVTX_READ);
    sprintf(filename, "%s/%s", dirname, argv[3]);
    if (!read_atm(filename, &ctl, atm))
      ERRMSG("Cannot open file!");
    RANGE_POP;

    /* 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;

    /* Update GPU... */
#ifdef _OPENACC
    RANGE_PUSH("Update device", NVTX_H2D);
#pragma acc update device(atm[:1],ctl)
    RANGE_POP;
#endif

    /* Initialize random number generator... */
    module_diffusion_init();

    /* Initialize meteorological data... */
    RANGE_PUSH("Init meteo data", NVTX_READ);
    get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
    if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
      WARN("Violation of CFL criterion! Check DT_MOD!");
    RANGE_POP;

    /* Initialize isosurface... */
    RANGE_PUSH("Init isosurface...", NVTX_CPU);
    if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
      module_isosurf_init(&ctl, met0, met1, atm, cache);
    RANGE_POP;

    /* Update GPU... */
#ifdef _OPENACC
    RANGE_PUSH("Update device", NVTX_H2D);
#pragma acc update device(cache[:1])
    RANGE_POP;
#endif

    /* 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... */
      RANGE_PUSH("Set time steps", NVTX_GPU);
#ifdef _OPENACC
#pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
#endif
      for (int ip = 0; ip < atm->np; ip++) {
        double atmtime = atm->time[ip];
        double tstart = ctl.t_start;
        double tstop = ctl.t_stop;
        int dir = ctl.direction;
        if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
             && dir * (atmtime - t) < 0))
```

```
        dt[ip] = t - atmtime;
      else
        dt[ip] = 0;
    }
    RANGE_POP;

    /* Get meteorological data... */
    RANGE_PUSH("Get meteo data", NVTX_READ);
    START_TIMER(TIMER_INPUT);
    if (t != ctl.t_start)
      get_met(&ctl, argv[4], t, &met0, &met1);
    STOP_TIMER(TIMER_INPUT);
    RANGE_POP;

    /* Check initial positions... */
    RANGE_PUSH("Check initial positions", NVTX_GPU);
    START_TIMER(TIMER_POSITION);
    module_position(met0, met1, atm, dt);
    STOP_TIMER(TIMER_POSITION);
    RANGE_POP;

    /* Advection... */
    RANGE_PUSH("Advection", NVTX_GPU);
    START_TIMER(TIMER_ADVECT);
    module_advection(met0, met1, atm, dt);
    STOP_TIMER(TIMER_ADVECT);
    RANGE_POP;

    /* Turbulent diffusion... */
    RANGE_PUSH("Turbulent diffusion", NVTX_GPU);
    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) {
      module_diffusion_rng(rs, 3 * (size_t) atm->np);
      module_diffusion_turb(&ctl, atm, dt, rs);
    }
    STOP_TIMER(TIMER_DIFFTURB);
    RANGE_POP;

    /* Mesoscale diffusion... */
    RANGE_PUSH("Mesoscale diffusion", NVTX_GPU);
    START_TIMER(TIMER_DIFFMESO);
    if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
      module_diffusion_rng(rs, 3 * (size_t) atm->np);
      module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
    }
    STOP_TIMER(TIMER_DIFFMESO);
    RANGE_POP;

    /* Sedimentation... */
    RANGE_PUSH("Sedimentation", NVTX_GPU);
    START_TIMER(TIMER_SEDI);
    if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
      module_sedi(&ctl, met0, met1, atm, dt);
    STOP_TIMER(TIMER_SEDI);
    RANGE_POP;

    /* Isosurface... */
    RANGE_PUSH("Isosurface", NVTX_GPU);
    START_TIMER(TIMER_ISOSURF);
    if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
      module_isosurf(&ctl, met0, met1, atm, cache);
    STOP_TIMER(TIMER_ISOSURF);
    RANGE_POP;

    /* Check final positions... */
    RANGE_PUSH("Check final positions", NVTX_GPU);
    START_TIMER(TIMER_POSITION);
    module_position(met0, met1, atm, dt);
    STOP_TIMER(TIMER_POSITION);
    RANGE_POP;

    /* Interpolate meteorological data... */
    RANGE_PUSH("Interpolate meteo data", NVTX_GPU);
    START_TIMER(TIMER_METEO);
    if (ctl.met_dt_out > 0
        && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0))
      module_meteo(&ctl, met0, met1, atm);
    STOP_TIMER(TIMER_METEO);
    RANGE_POP;

    /* Decay of particle mass... */
    RANGE_PUSH("Decay of particle mass", NVTX_GPU);
    START_TIMER(TIMER_DECAY);
    if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
      module_decay(&ctl, atm, dt);
```

```
      STOP_TIMER(TIMER_DECAY);
      RANGE_POP;

      /* OH chemistry... */
      RANGE_PUSH("OH chem", NVTX_GPU);
      START_TIMER(TIMER_OHCHEM);
      if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
        module_oh_chem(&ctl, met0, met1, atm, dt);
      STOP_TIMER(TIMER_OHCHEM);
      RANGE_POP;

      /* Wet deposition... */
      RANGE_PUSH("Wet deposition", NVTX_GPU);
      START_TIMER(TIMER_WETDEPO);
      if (ctl.wet_depo[0] > 0 && ctl.wet_depo[1] > 0
          && ctl.wet_depo[2] > 0 && ctl.wet_depo[3] > 0)
        module_wet_deposition(&ctl, met0, met1, atm, dt);
      STOP_TIMER(TIMER_WETDEPO);
      RANGE_POP;

      /* Write output... */
      RANGE_PUSH("Write output", NVTX_WRITE);
      START_TIMER(TIMER_OUTPUT);
      write_output(dirname, &ctl, met0, met1, atm, t);
      STOP_TIMER(TIMER_OUTPUT);
      RANGE_POP;
    }

    /* ------------------------------------------------------------
       Finalize model run...
       ------------------------------------------------------------ */

    /* Report problem size... */
    printf("SIZE_NP = %d\n", atm->np);
    printf("SIZE_MPI_TASKS = %d\n", size);
    printf("SIZE_OMP_THREADS = %d\n", omp_get_max_threads());
    printf("SIZE_ACC_DEVICES = %d\n", num_devices);

    /* Report memory usage... */
    printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
    printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
    printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
    printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
                                           + 4 * NP * sizeof(double)
                                           + EX * EY * EP * sizeof(float)) /
           1024. / 1024.);
    printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
                                          + EX * EY * EP * sizeof(float)
                                          + 4 * GX * GY * GZ * sizeof(double)
                                          + 2 * GX * GY * GZ * sizeof(int)
                                          + 2 * GX * GY * sizeof(double)
                                          + GX * GY * sizeof(int)) / 1024. /
           1024.);

    /* Report timers... */
    STOP_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);
    PRINT_TIMER(TIMER_OHCHEM);
    PRINT_TIMER(TIMER_WETDEPO);
    PRINT_TIMER(TIMER_TOTAL);

    /* Free... */
    RANGE_PUSH("Deallocations", NVTX_CPU);
    free(atm);
    free(cache);
    free(met0);
    free(met1);
    free(dt);
    free(rs);
    RANGE_POP;

    /* Delete data region on GPUs... */
#ifdef _OPENACC
    RANGE_PUSH("Delete data region", NVTX_GPU);
#pragma acc exit data delete(ctl,atm,cache,met0,met1,dt,rs)
    RANGE_POP;
#endif
```

```
  }

  /* Finalize MPI... */
#ifdef MPI
  RANGE_PUSH("Finalize MPI", NVTX_CPU);
  MPI_Finalize();
  RANGE_POP;
#endif

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.33.3 Variable Documentation

5.33.3.1 static gsl_rng ∗ rng

Definition at line 32 of file trac.c.

5.34 trac.c

```
00001 /*
00002   This file is part of MPTRAC.
00003 
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008 
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013 
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016 
00017   Copyright (C) 2013-2020 Forschungszentrum Juelich GmbH
00018 */
00019 
00025 #include "libtrac.h"
00026 
00027 /* ------------------------------------------------------------
00028    Global variables...
00029    ------------------------------------------------------------ */
00030 
00031 #ifdef _OPENACC
00032 curandGenerator_t rng;
00033 #else
00034 static gsl_rng *rng[NTHREADS];
00035 #endif
00036 
00037 /* ------------------------------------------------------------
00038    Functions...
00039    ------------------------------------------------------------ */
00040 
00042 void module_advection(
00043   met_t * met0,
00044   met_t * met1,
00045   atm_t * atm,
00046   double *dt);
00047 
00049 void module_decay(
00050   ctl_t * ctl,
00051   atm_t * atm,
00052   double *dt);
00053 
00055 void module_diffusion_init(
00056   void);
00057 
00059 void module_diffusion_meso(
00060   ctl_t * ctl,
00061   met_t * met0,
00062   met_t * met1,
00063   atm_t * atm,
00064   cache_t * cache,
00065   double *dt,
00066   double *rs);
00067 
00069 void module_diffusion_rng(
00070   double *rs,
00071   size_t n);
00072 
00074 void module_diffusion_turb(
00075   ctl_t * ctl,
00076   atm_t * atm,
00077   double *dt,
00078   double *rs);
00079 
00081 void module_isosurf_init(
00082   ctl_t * ctl,
00083   met_t * met0,
```

```
  met_t * met1,
  atm_t * atm,
  cache_t * cache);

void module_isosurf(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  cache_t * cache);

void module_meteo(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm);

void module_position(
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_sedi(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_oh_chem(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_wet_deposition(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  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;

  cache_t *cache;

  met_t *met0, *met1;

  FILE *dirlist;

  char dirname[LEN], filename[2 * LEN];

  double *dt, *rs, t;

  int num_devices = 0, ntask = -1, rank = 0, size = 1;

  /* Initialize MPI... */
#ifdef MPI
  RANGE_PUSH("Initialize MPI", NVTX_CPU);
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  RANGE_POP;
#endif

  /* Initialize GPUs... */
#ifdef _OPENACC
```

```
  RANGE_PUSH("Initialize GPUs", NVTX_GPU);
  acc_device_t device_type = acc_get_device_type();
  num_devices = acc_get_num_devices(acc_device_nvidia);
  int device_num = rank % num_devices;
  acc_set_device_num(device_num, acc_device_nvidia);
  acc_init(device_type);
  RANGE_POP;
#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... */
    RANGE_PUSH("Allocate", NVTX_CPU);
    ALLOC(atm, atm_t, 1);
    ALLOC(cache, cache_t, 1);
    ALLOC(met0, met_t, 1);
    ALLOC(met1, met_t, 1);
    ALLOC(dt, double,
          NP);
    ALLOC(rs, double,
          3 * NP);
    RANGE_POP;

    /* Create data region on GPUs... */
#ifdef _OPENACC
    RANGE_PUSH("Create data region", NVTX_GPU);
#pragma acc enter data create(atm[:1],cache[:1],ctl,met0[:1],met1[:1],dt[:NP],rs[:3*NP])
    RANGE_POP;
#endif

    /* Read control parameters... */
    RANGE_PUSH("Read ctl", NVTX_READ);
    sprintf(filename, "%s/%s", dirname, argv[2]);
    read_ctl(filename, argc, argv, &ctl);
    RANGE_POP;

    /* Read atmospheric data... */
    RANGE_PUSH("Read atm", NVTX_READ);
    sprintf(filename, "%s/%s", dirname, argv[3]);
    if (!read_atm(filename, &ctl, atm))
      ERRMSG("Cannot open file!");
    RANGE_POP;

    /* 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;

```

```
    /* Update GPU... */
#ifdef _OPENACC
    RANGE_PUSH("Update device", NVTX_H2D);
#pragma acc update device(atm[:1],ctl)
    RANGE_POP;
#endif

    /* Initialize random number generator... */
    module_diffusion_init();

    /* Initialize meteorological data... */
    RANGE_PUSH("Init meteo data", NVTX_READ);
    get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
    if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
      WARN("Violation of CFL criterion! Check DT_MOD!");
    RANGE_POP;

    /* Initialize isosurface... */
    RANGE_PUSH("Init isosurface...", NVTX_CPU);
    if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
      module_isosurf_init(&ctl, met0, met1, atm, cache);
    RANGE_POP;

    /* Update GPU... */
#ifdef _OPENACC
    RANGE_PUSH("Update device", NVTX_H2D);
#pragma acc update device(cache[:1])
    RANGE_POP;
#endif

    /* 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... */
      RANGE_PUSH("Set time steps", NVTX_GPU);
#ifdef _OPENACC
#pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
#endif
      for (int ip = 0; ip < atm->np; ip++) {
        double atmtime = atm->time[ip];
        double tstart = ctl.t_start;
        double tstop = ctl.t_stop;
        int dir = ctl.direction;
        if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
             && dir * (atmtime - t) < 0))
          dt[ip] = t - atmtime;
        else
          dt[ip] = 0;
      }
      RANGE_POP;

      /* Get meteorological data... */
      RANGE_PUSH("Get meteo data", NVTX_READ);
      START_TIMER(TIMER_INPUT);
      if (t != ctl.t_start)
        get_met(&ctl, argv[4], t, &met0, &met1);
      STOP_TIMER(TIMER_INPUT);
      RANGE_POP;

      /* Check initial positions... */
      RANGE_PUSH("Check initial positions", NVTX_GPU);
      START_TIMER(TIMER_POSITION);
      module_position(met0, met1, atm, dt);
      STOP_TIMER(TIMER_POSITION);
      RANGE_POP;

      /* Advection... */
      RANGE_PUSH("Advection", NVTX_GPU);
      START_TIMER(TIMER_ADVECT);
      module_advection(met0, met1, atm, dt);
      STOP_TIMER(TIMER_ADVECT);
      RANGE_POP;

      /* Turbulent diffusion... */
```

```
    RANGE_PUSH("Turbulent diffusion", NVTX_GPU);
    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) {
      module_diffusion_rng(rs, 3 * (size_t) atm->np);
      module_diffusion_turb(&ctl, atm, dt, rs);
    }
    STOP_TIMER(TIMER_DIFFTURB);
    RANGE_POP;

    /* Mesoscale diffusion... */
    RANGE_PUSH("Mesoscale diffusion", NVTX_GPU);
    START_TIMER(TIMER_DIFFMESO);
    if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
      module_diffusion_rng(rs, 3 * (size_t) atm->np);
      module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
    }
    STOP_TIMER(TIMER_DIFFMESO);
    RANGE_POP;

    /* Sedimentation... */
    RANGE_PUSH("Sedimentation", NVTX_GPU);
    START_TIMER(TIMER_SEDI);
    if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
      module_sedi(&ctl, met0, met1, atm, dt);
    STOP_TIMER(TIMER_SEDI);
    RANGE_POP;

    /* Isosurface... */
    RANGE_PUSH("Isosurface", NVTX_GPU);
    START_TIMER(TIMER_ISOSURF);
    if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
      module_isosurf(&ctl, met0, met1, atm, cache);
    STOP_TIMER(TIMER_ISOSURF);
    RANGE_POP;

    /* Check final positions... */
    RANGE_PUSH("Check final positions", NVTX_GPU);
    START_TIMER(TIMER_POSITION);
    module_position(met0, met1, atm, dt);
    STOP_TIMER(TIMER_POSITION);
    RANGE_POP;

    /* Interpolate meteorological data... */
    RANGE_PUSH("Interpolate meteo data", NVTX_GPU);
    START_TIMER(TIMER_METEO);
    if (ctl.met_dt_out > 0
        && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
met_dt_out) == 0))
      module_meteo(&ctl, met0, met1, atm);
    STOP_TIMER(TIMER_METEO);
    RANGE_POP;

    /* Decay of particle mass... */
    RANGE_PUSH("Decay of particle mass", NVTX_GPU);
    START_TIMER(TIMER_DECAY);
    if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
      module_decay(&ctl, atm, dt);
    STOP_TIMER(TIMER_DECAY);
    RANGE_POP;

    /* OH chemistry... */
    RANGE_PUSH("OH chem", NVTX_GPU);
    START_TIMER(TIMER_OHCHEM);
    if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
      module_oh_chem(&ctl, met0, met1, atm, dt);
    STOP_TIMER(TIMER_OHCHEM);
    RANGE_POP;

    /* Wet deposition... */
    RANGE_PUSH("Wet deposition", NVTX_GPU);
    START_TIMER(TIMER_WETDEPO);
    if (ctl.wet_depo[0] > 0 && ctl.wet_depo[1] > 0
        && ctl.wet_depo[2] > 0 && ctl.wet_depo[3] > 0)
      module_wet_deposition(&ctl, met0, met1, atm, dt);
    STOP_TIMER(TIMER_WETDEPO);
    RANGE_POP;

    /* Write output... */
    RANGE_PUSH("Write output", NVTX_WRITE);
    START_TIMER(TIMER_OUTPUT);
    write_output(dirname, &ctl, met0, met1, atm, t);
    STOP_TIMER(TIMER_OUTPUT);
    RANGE_POP;
  }

  /* ------------------------------------------------------------
```

```
       Finalize model run...
       ------------------------------------------------------------ */

    /* Report problem size... */
    printf("SIZE_NP = %d\n", atm->np);
    printf("SIZE_MPI_TASKS = %d\n", size);
    printf("SIZE_OMP_THREADS = %d\n", omp_get_max_threads());
    printf("SIZE_ACC_DEVICES = %d\n", num_devices);

    /* Report memory usage... */
    printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
    printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
    printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
    printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
                                           + 4 * NP * sizeof(double)
                                           + EX * EY * EP * sizeof(float)) /
           1024. / 1024.);
    printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
                                          + EX * EY * EP * sizeof(float)
                                          + 4 * GX * GY * GZ * sizeof(double)
                                          + 2 * GX * GY * GZ * sizeof(int)
                                          + 2 * GX * GY * sizeof(double)
                                          + GX * GY * sizeof(int)) / 1024. /
           1024.);

    /* Report timers... */
    STOP_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);
    PRINT_TIMER(TIMER_OHCHEM);
    PRINT_TIMER(TIMER_WETDEPO);
    PRINT_TIMER(TIMER_TOTAL);

    /* Free... */
    RANGE_PUSH("Deallocations", NVTX_CPU);
    free(atm);
    free(cache);
    free(met0);
    free(met1);
    free(dt);
    free(rs);
    RANGE_POP;

    /* Delete data region on GPUs... */
#ifdef _OPENACC
    RANGE_PUSH("Delete data region", NVTX_GPU);
#pragma acc exit data delete(ctl,atm,cache,met0,met1,dt,rs)
    RANGE_POP;
#endif
  }

  /* Finalize MPI... */
#ifdef MPI
  RANGE_PUSH("Finalize MPI", NVTX_CPU);
  MPI_Finalize();
  RANGE_POP;
#endif

  return EXIT_SUCCESS;
}

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

void module_advection(
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

#ifdef _OPENACC
#pragma acc data present(met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

```

```
      int ci[3] = { 0 };

      double dtm = 0.0, v[3] = { 0.0 }, xm[3] = {
      0.0};
      double cw[3] = { 0.0 };

      /* Interpolate meteorological data... */
      intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[0], ci,
                         cw, 1);
      intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[1], ci,
                         cw, 0);
      intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &v[2], ci,
                         cw, 0);

      /* Get position of the mid point... */
      dtm = atm->time[ip] + 0.5 * dt[ip];
      xm[0] =
        atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
      xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
      xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];

      /* Interpolate meteorological data for mid point... */
      intpol_met_time_3d(met0, met0->u, met1, met1->u, dtm, xm[2], xm[0],
                         xm[1], &v[0], ci, cw, 1);
      intpol_met_time_3d(met0, met0->v, met1, met1->v, dtm, xm[2], xm[0],
                         xm[1], &v[1], ci, cw, 0);
      intpol_met_time_3d(met0, met0->w, met1, met1->w, dtm, xm[2], xm[0],
                         xm[1], &v[2], ci, cw, 0);

      /* Save new position... */
      atm->time[ip] += dt[ip];
      atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
      atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
      atm->p[ip] += dt[ip] * v[2];
    }
}

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

void module_decay(
  ctl_t * ctl,
  atm_t * atm,
  double *dt) {

  /* Check quantity flags... */
  if (ctl->qnt_m < 0)
    ERRMSG("Module needs quantity mass!");

#ifdef _OPENACC
#pragma acc data present(ctl,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double p0, p1, pt, tdec, 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 lifetime... */
      tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;

      /* Calculate exponential decay... */
      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
    }
}

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

```

void module_diffusion_init(
  void) {

  /* Initialize random number generator... */
#ifdef _OPENACC

  if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
      != CURAND_STATUS_SUCCESS)
    ERRMSG("Cannot create random number generator!");
  if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
      != CURAND_STATUS_SUCCESS)
    ERRMSG("Cannot set stream for random number generator!");

#else

  gsl_rng_env_setup();
  if (omp_get_max_threads() > NTHREADS)
    ERRMSG("Too many threads!");
  for (int 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);
  }

#endif
}

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

void module_diffusion_meso(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  cache_t * cache,
  double *dt,
  double *rs) {

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double u[16], v[16], w[16];

      /* Get indices... */
      int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
      int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
      int iz = locate_irr(met0->p, met0->np, atm->p[ip]);

      /* Caching of wind standard deviations... */
      if (cache->tsig[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... */
        cache->usig[ix][iy][iz] = (float) stddev(u, 16);
        cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
        cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
        cache->tsig[ix][iy][iz] = met0->time;
      }

      /* Set temporal correlations for mesoscale fluctuations... */
      double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
      double r2 = sqrt(1 - r * r);

      /* Calculate horizontal mesoscale wind fluctuations... */
      if (ctl->turb_mesox > 0) {
        cache->up[ip] = (float)
          (r * cache->up[ip]
           + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
        atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);

        cache->vp[ip] = (float)
          (r * cache->vp[ip]
           + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
        atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
      }

      /* Calculate vertical mesoscale wind fluctuations... */
      if (ctl->turb_mesoz > 0) {
        cache->wp[ip] = (float)
          (r * cache->wp[ip]
           + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
        atm->p[ip] += cache->wp[ip] * dt[ip];
      }
    }
}

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

void module_diffusion_rng(
  double *rs,
  size_t n) {

#ifdef _OPENACC

#pragma acc host_data use_device(rs)
  {
    if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
        != CURAND_STATUS_SUCCESS)
      ERRMSG("Cannot create random numbers!");
  }

#else

#pragma omp parallel for default(shared)
  for (size_t i = 0; i < n; ++i)
    rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);

#endif

}

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

```
void module_diffusion_turb(
  ctl_t * ctl,
  atm_t * atm,
  double *dt,
  double *rs) {

#ifdef _OPENACC
#pragma acc data present(ctl,atm,dt,rs)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double w;

      /* Get tropopause pressure... */
      double pt = clim_tropo(atm->time[ip], atm->lat[ip]);

      /* Get weighting factor... */
      double p1 = pt * 0.866877899;
      double 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... */
      double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
turb_dx_strat;
      double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
turb_dz_strat;

      /* Horizontal turbulent diffusion... */
      if (dx > 0) {
        double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
        atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
        atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
      }

      /* Vertical turbulent diffusion... */
      if (dz > 0) {
        double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
        atm->p[ip]
          += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
      }
    }
}

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

void module_isosurf_init(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  cache_t * cache) {

  FILE *in;

  char line[LEN];

  double t, cw[3];

  int ci[3];

  /* Save pressure... */
  if (ctl->isosurf == 1)
    for (int ip = 0; ip < atm->np; ip++)
      cache->iso_var[ip] = atm->p[ip];

  /* Save density... */
  else if (ctl->isosurf == 2)
    for (int ip = 0; ip < atm->np; ip++) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                         1);
      cache->iso_var[ip] = atm->p[ip] / t;
    }

  /* Save potential temperature... */
  else if (ctl->isosurf == 3)
    for (int ip = 0; ip < atm->np; ip++) {
```

```
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                          atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                          1);
      cache->iso_var[ip] = THETA(atm->p[ip], 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", &(cache->iso_ts[cache->iso_n]),
                 &(cache->iso_ps[cache->iso_n])) == 2)
        if ((++cache->iso_n) > NP)
          ERRMSG("Too many data points!");

    /* Check number of points... */
    if (cache->iso_n < 1)
      ERRMSG("Could not read any data!");

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

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

void module_isosurf(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  cache_t * cache) {

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,cache)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++) {

    double t, cw[3];

    int ci[3];

    /* Restore pressure... */
    if (ctl->isosurf == 1)
      atm->p[ip] = cache->iso_var[ip];

    /* Restore density... */
    else if (ctl->isosurf == 2) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                          atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                          1);
      atm->p[ip] = cache->iso_var[ip] * t;
    }

    /* Restore potential temperature... */
    else if (ctl->isosurf == 3) {
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                          atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
                          1);
      atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
    }

    /* Interpolate pressure... */
    else if (ctl->isosurf == 4) {
      if (atm->time[ip] <= cache->iso_ts[0])
        atm->p[ip] = cache->iso_ps[0];
      else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
        atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
      else {
        int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
        atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
                          cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
```

```
                              atm->time[ip]);
        }
      }
    }
}

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

void module_meteo(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm) {

  /* Check quantity flags... */
  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!");

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++) {

    double ps, pt, pc, pv, t, u, v, w, h2o, o3, lwc, iwc, z, cw[3];

    int ci[3];

    /* Interpolate meteorological data... */
    intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
    intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
    intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
    intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
    intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
    intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &pv, ci, cw,
                       0);
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &h2o, ci, cw,
                       0);
    intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &o3, ci, cw,
                       0);
    intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
                       0);
    intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->time[ip],
                       atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
                       0);
    intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
    intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
    intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->time[ip],
                       atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);

    /* 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;

  /* Set cloud liquid water content... */
  if (ctl->qnt_lwc >= 0)
    atm->q[ctl->qnt_lwc][ip] = lwc;

  /* Set cloud ice water content... */
  if (ctl->qnt_iwc >= 0)
    atm->q[ctl->qnt_iwc][ip] = iwc;

  /* Set cloud top pressure... */
  if (ctl->qnt_pc >= 0)
    atm->q[ctl->qnt_pc][ip] = pc;

  /* Set nitric acid vmr... */
  if (ctl->qnt_hno3 >= 0)
    atm->q[ctl->qnt_hno3][ip] =
      clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]);

  /* Set hydroxyl number concentration... */
  if (ctl->qnt_oh >= 0)
    atm->q[ctl->qnt_oh][ip] =
      clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]);

  /* 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 relative humidty... */
  if (ctl->qnt_rh >= 0)
    atm->q[ctl->qnt_rh][ip] = RH(atm->p[ip], t, h2o);

  /* 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) {
    double p_hno3;
    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;
```

```
      double p_h2o =
        (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
      double a = 0.009179 - 0.00088 * log10(p_h2o);
      double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
      double c = -11397.0 / a;
      double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
      double 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)
      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,
  double *dt) {

#ifdef _OPENACC
#pragma acc data present(met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double ps, cw[3];

      int ci[3];

      /* 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;

      /* Check pressure... */
      if (atm->p[ip] < met0->p[met0->np - 1])
        atm->p[ip] = met0->p[met0->np - 1];
      else if (atm->p[ip] > 300.) {
        intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
                           atm->lon[ip], atm->lat[ip], &ps, ci, cw, 1);
        if (atm->p[ip] > ps)
          atm->p[ip] = ps;
      }
    }
}

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

void module_sedi(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,dt)
```

```
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p, cw[3];

      int ci[3];

      /* 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_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
                         1);

      /* 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 * 4.8096e-26));

      /* 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 * (1.249 + 0.42 * exp(-0.87 / 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[ip] / 1000., atm->p[ip]);
    }
}

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

void module_oh_chem(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Check quantity flags... */
  if (ctl->qnt_m < 0)
    ERRMSG("Module needs quantity mass!");

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double c, k, k0, ki, M, T, cw[3];

      int ci[3];

      /* Get temperature... */
      intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
                         1);

      /* Calculate molecular density... */
      M = 7.243e21 * (atm->p[ip] / P0) / T;

      /* Calculate rate coefficient for X + OH + M -> XOH + M
         (JPL Publication 15-10) ... */
      k0 = ctl->oh_chem[0] *
        (ctl->oh_chem[1] > 0 ? pow(T / 300., -ctl->oh_chem[1]) : 1.);
```

```
      ki = ctl->oh_chem[2] *
        (ctl->oh_chem[3] > 0 ? pow(T / 300., -ctl->oh_chem[3]) : 1.);
      c = log10(k0 * M / ki);
      k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));

      /* Calculate exponential decay... */
      atm->q[ctl->qnt_m][ip] *=
        exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
    }
}

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

void module_wet_deposition(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Check quantity flags... */
  if (ctl->qnt_m < 0)
    ERRMSG("Module needs quantity mass!");

#ifdef _OPENACC
#pragma acc data present(ctl,met0,met1,atm,dt)
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < atm->np; ip++)
    if (dt[ip] != 0) {

      double H, Is, Si, T, cl, lambda, iwc, lwc, pc, cw[3];

      int inside, ci[3];

      /* Check whether particle is below cloud top... */
      intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->time[ip],
                         atm->lon[ip], atm->lat[ip], &pc, ci, cw, 1);
      if (!isfinite(pc) || atm->p[ip] <= pc)
        continue;

      /* Check whether particle is inside or below cloud... */
      intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
                         1);
      intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->time[ip],
                         atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
                         0);
      inside = (iwc > 0 || lwc > 0);

      /* Estimate precipitation rate (Pisso et al., 2019)... */
      intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->time[ip],
                         atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
      Is = pow(2. * cl, 1. / 0.36);
      if (Is < 0.01)
        continue;

      /* Calculate in-cloud scavenging for gases... */
      if (inside) {

        /* Get temperature... */
        intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                           atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
                           0);

        /* Get Henry's constant (Sander, 2015)... */
        H = ctl->wet_depo[2] * 101.325
          * exp(ctl->wet_depo[3] * (1. / T - 1. / 298.15));

        /* Get scavenging coefficient (Hertel et al., 1995)... */
        Si = 1. / ((1. - cl) / (H * RI / P0 * T) + cl);
        lambda = 6.2 * Si * Is / 3.6e6;
      }

      /* Calculate below-cloud scavenging for gases (Pisso et al., 2019)... */
      else
        lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);

      /* Calculate exponential decay... */
```

```
      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
    }
}

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

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, updated = 0;

  /* 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) {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write atm data", NVTX_WRITE);
    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);
    RANGE_POP;
  }

  /* Write gridded data... */
  if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write grid data", NVTX_WRITE);
    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);
    RANGE_POP;
  }

  /* Write CSI data... */
  if (ctl->csi_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write CSI data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
    write_csi(filename, ctl, atm, t);
    RANGE_POP;
  }

  /* Write ensemble data... */
  if (ctl->ens_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write ensemble data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
    write_ens(filename, ctl, atm, t);
    RANGE_POP;
```

```
  }

  /* Write profile data... */
  if (ctl->prof_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write profile data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write_prof(filename, ctl, met0, met1, atm, t);
    RANGE_POP;
  }

  /* Write station data... */
  if (ctl->stat_basename[0] != '-') {
    if (!updated) {
      RANGE_PUSH("Update host", NVTX_D2H);
#ifdef _OPENACC
#pragma acc update host(atm[:1])
#endif
      RANGE_POP;
      updated = 1;
    }
    RANGE_PUSH("Write station data", NVTX_WRITE);
    sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
    write_station(filename, ctl, atm, t);
    RANGE_POP;
  }
}
```

5.35 tropo.c File Reference

Create tropopause climatology from meteorological data.

Functions

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

5.35.1 Detailed Description

Create tropopause climatology from meteorological data.

Definition in file tropo.c.

5.35.2 Function Documentation

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

Definition at line 337 of file tropo.c.

```
                {

  int varid;

  NC(nc_inq_varid(ncid, varname, &varid));
  NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
}
```

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

Definition at line 41 of file tropo.c.

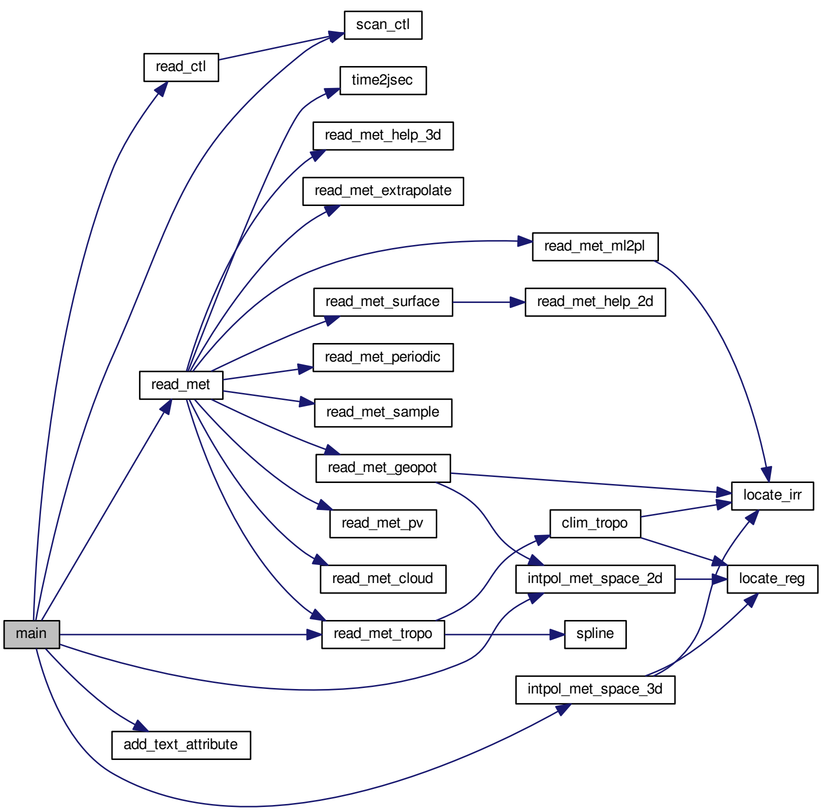
```
{

  ctl_t ctl;

  met_t *met;

  static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
    lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];

  static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
    clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmo1pid,
    wmo1qid, wmo1tid, wmo1zid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];

  static size_t count[10], start[10];

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

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
  h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);

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

    /* Read meteorological data... */
    ctl.met_tropo = 0;
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Set horizontal grid... */
    if (!init) {
      init = 1;

      /* Get 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!");
      }

      /* Create netCDF file... */
      printf("Write tropopause data file: %s\n", argv[2]);
      NC(nc_create(argv[2], NC_CLOBBER, &ncid));

      /* Create dimensions... */
      NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
      NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
      NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));

      /* Create variables... */
      NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
```

```
    NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
    NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
    NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
    NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
    NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
    if (h2o)
      NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
    NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
    NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
    NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
    if (h2o)
      NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
    NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
    NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
    NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
    if (h2o)
      NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
    NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
    NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
    NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
    if (h2o)
      NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));

    /* Set attributes... */
    add_text_attribute(ncid, "time", "units",
                       "seconds since 2000-01-01 00:00:00 UTC");
    add_text_attribute(ncid, "time", "long_name", "time");
    add_text_attribute(ncid, "lon", "units", "degrees_east");
    add_text_attribute(ncid, "lon", "long_name", "longitude");
    add_text_attribute(ncid, "lat", "units", "degrees_north");
    add_text_attribute(ncid, "lat", "long_name", "latitude");

    add_text_attribute(ncid, "clp_z", "units", "km");
    add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
    add_text_attribute(ncid, "clp_p", "units", "hPa");
    add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
    add_text_attribute(ncid, "clp_t", "units", "K");
    add_text_attribute(ncid, "clp_t", "long_name",
                       "cold point temperature");
    if (h2o) {
      add_text_attribute(ncid, "clp_q", "units", "ppv");
      add_text_attribute(ncid, "clp_q", "long_name",
                         "cold point water vapor");
    }

    add_text_attribute(ncid, "dyn_z", "units", "km");
    add_text_attribute(ncid, "dyn_z", "long_name",
                       "dynamical tropopause height");
    add_text_attribute(ncid, "dyn_p", "units", "hPa");
    add_text_attribute(ncid, "dyn_p", "long_name",
                       "dynamical tropopause pressure");
    add_text_attribute(ncid, "dyn_t", "units", "K");
    add_text_attribute(ncid, "dyn_t", "long_name",
                       "dynamical tropopause temperature");
    if (h2o) {
      add_text_attribute(ncid, "dyn_q", "units", "ppv");
      add_text_attribute(ncid, "dyn_q", "long_name",
                         "dynamical tropopause water vapor");
    }

    add_text_attribute(ncid, "wmo_1st_z", "units", "km");
    add_text_attribute(ncid, "wmo_1st_z", "long_name",
                       "WMO 1st tropopause height");
    add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
    add_text_attribute(ncid, "wmo_1st_p", "long_name",
                       "WMO 1st tropopause pressure");
    add_text_attribute(ncid, "wmo_1st_t", "units", "K");
    add_text_attribute(ncid, "wmo_1st_t", "long_name",
                       "WMO 1st tropopause temperature");
    if (h2o) {
      add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
      add_text_attribute(ncid, "wmo_1st_q", "long_name",
                         "WMO 1st tropopause water vapor");
    }

    add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
    add_text_attribute(ncid, "wmo_2nd_z", "long_name",
                       "WMO 2nd tropopause height");
    add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
    add_text_attribute(ncid, "wmo_2nd_p", "long_name",
                       "WMO 2nd tropopause pressure");
    add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
    add_text_attribute(ncid, "wmo_2nd_t", "long_name",
                       "WMO 2nd tropopause temperature");
    if (h2o) {
      add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
      add_text_attribute(ncid, "wmo_2nd_q", "long_name",
```

```
                           "WMO 2nd tropopause water vapor");
      }

      /* End definition... */
      NC(nc_enddef(ncid));

      /* Write longitude and latitude... */
      NC(nc_put_var_double(ncid, latid, lats));
      NC(nc_put_var_double(ncid, lonid, lons));
    }

    /* Write time... */
    start[0] = (size_t) nt;
    count[0] = 1;
    start[1] = 0;
    count[1] = (size_t) ny;
    start[2] = 0;
    count[2] = (size_t) nx;
    NC(nc_put_vara_double(ncid, timid, start, count, &met->time));

    /* Get cold point... */
    ctl.met_tropo = 2;
    read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                            &pt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                            lats[iy], &zt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                            lats[iy], &tt[iy * nx + ix], ci, cw, 0);
        intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                            lats[iy], &qt[iy * nx + ix], ci, cw, 0);
      }

    /* Write data... */
    NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
    NC(nc_put_vara_double(ncid, clppid, start, count, pt));
    NC(nc_put_vara_double(ncid, clptid, start, count, tt));
    if (h2o)
      NC(nc_put_vara_double(ncid, clpqid, start, count, qt));

    /* Get dynamical tropopause... */
    ctl.met_tropo = 5;
    read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                            &pt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                            lats[iy], &zt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                            lats[iy], &tt[iy * nx + ix], ci, cw, 0);
        intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                            lats[iy], &qt[iy * nx + ix], ci, cw, 0);
      }

    /* Write data... */
    NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
    NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
    NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
    if (h2o)
      NC(nc_put_vara_double(ncid, dynqid, start, count, qt));

    /* Get WMO 1st tropopause... */
    ctl.met_tropo = 3;
    read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                            &pt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                            lats[iy], &zt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                            lats[iy], &tt[iy * nx + ix], ci, cw, 0);
        intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                            lats[iy], &qt[iy * nx + ix], ci, cw, 0);
      }

    /* Write data... */
    NC(nc_put_vara_double(ncid, wmo1zid, start, count, zt));
    NC(nc_put_vara_double(ncid, wmo1pid, start, count, pt));
    NC(nc_put_vara_double(ncid, wmo1tid, start, count, tt));
    if (h2o)
```

```
      NC(nc_put_vara_double(ncid, wmo1qid, start, count, qt));

    /* Get WMO 2nd tropopause... */
    ctl.met_tropo = 4;
    read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                            &pt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                            lats[iy], &zt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                            lats[iy], &tt[iy * nx + ix], ci, cw, 0);
        intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                            lats[iy], &qt[iy * nx + ix], ci, cw, 0);
      }

    /* Write data... */
    NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
    NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
    NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
    if (h2o)
      NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));

    /* Increment time step counter... */
    nt++;
  }

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.36 tropo.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"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

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

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

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

  ctl_t ctl;

  met_t *met;

  static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
    lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];

  static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
    clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmo1pid,
    wmo1qid, wmo1tid, wmo1zid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];

  static size_t count[10], start[10];

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

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
  dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
  lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
  dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
  h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);

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

    /* Read meteorological data... */
    ctl.met_tropo = 0;
    if (!read_met(&ctl, argv[i], met))
      continue;

    /* Set horizontal grid... */
    if (!init) {
      init = 1;

      /* Get 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!");
    }

    /* Create netCDF file... */
    printf("Write tropopause data file: %s\n", argv[2]);
    NC(nc_create(argv[2], NC_CLOBBER, &ncid));

    /* Create dimensions... */
    NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
    NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
    NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));

    /* Create variables... */
    NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
    NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
    NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
    NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
    NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
    NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
    if (h2o)
      NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
    NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
    NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
    NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
    if (h2o)
      NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
    NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
    NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
    NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
    if (h2o)
      NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
    NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
    NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
    NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
    if (h2o)
      NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));

    /* Set attributes... */
    add_text_attribute(ncid, "time", "units",
                       "seconds since 2000-01-01 00:00:00 UTC");
    add_text_attribute(ncid, "time", "long_name", "time");
    add_text_attribute(ncid, "lon", "units", "degrees_east");
    add_text_attribute(ncid, "lon", "long_name", "longitude");
    add_text_attribute(ncid, "lat", "units", "degrees_north");
    add_text_attribute(ncid, "lat", "long_name", "latitude");

    add_text_attribute(ncid, "clp_z", "units", "km");
    add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
    add_text_attribute(ncid, "clp_p", "units", "hPa");
    add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
    add_text_attribute(ncid, "clp_t", "units", "K");
    add_text_attribute(ncid, "clp_t", "long_name",
                       "cold point temperature");
    if (h2o) {
      add_text_attribute(ncid, "clp_q", "units", "ppv");
      add_text_attribute(ncid, "clp_q", "long_name",
                         "cold point water vapor");
    }

    add_text_attribute(ncid, "dyn_z", "units", "km");
    add_text_attribute(ncid, "dyn_z", "long_name",
                       "dynamical tropopause height");
    add_text_attribute(ncid, "dyn_p", "units", "hPa");
    add_text_attribute(ncid, "dyn_p", "long_name",
                       "dynamical tropopause pressure");
    add_text_attribute(ncid, "dyn_t", "units", "K");
    add_text_attribute(ncid, "dyn_t", "long_name",
                       "dynamical tropopause temperature");
```

```
      if (h2o) {
        add_text_attribute(ncid, "dyn_q", "units", "ppv");
        add_text_attribute(ncid, "dyn_q", "long_name",
                           "dynamical tropopause water vapor");
      }

      add_text_attribute(ncid, "wmo_1st_z", "units", "km");
      add_text_attribute(ncid, "wmo_1st_z", "long_name",
                         "WMO 1st tropopause height");
      add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
      add_text_attribute(ncid, "wmo_1st_p", "long_name",
                         "WMO 1st tropopause pressure");
      add_text_attribute(ncid, "wmo_1st_t", "units", "K");
      add_text_attribute(ncid, "wmo_1st_t", "long_name",
                         "WMO 1st tropopause temperature");
      if (h2o) {
        add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
        add_text_attribute(ncid, "wmo_1st_q", "long_name",
                           "WMO 1st tropopause water vapor");
      }

      add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
      add_text_attribute(ncid, "wmo_2nd_z", "long_name",
                         "WMO 2nd tropopause height");
      add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
      add_text_attribute(ncid, "wmo_2nd_p", "long_name",
                         "WMO 2nd tropopause pressure");
      add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
      add_text_attribute(ncid, "wmo_2nd_t", "long_name",
                         "WMO 2nd tropopause temperature");
      if (h2o) {
        add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
        add_text_attribute(ncid, "wmo_2nd_q", "long_name",
                           "WMO 2nd tropopause water vapor");
      }

      /* End definition... */
      NC(nc_enddef(ncid));

      /* Write longitude and latitude... */
      NC(nc_put_var_double(ncid, latid, lats));
      NC(nc_put_var_double(ncid, lonid, lons));
    }

    /* Write time... */
    start[0] = (size_t) nt;
    count[0] = 1;
    start[1] = 0;
    count[1] = (size_t) ny;
    start[2] = 0;
    count[2] = (size_t) nx;
    NC(nc_put_vara_double(ncid, timid, start, count, &met->time));

    /* Get cold point... */
    ctl.met_tropo = 2;
    read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                            &pt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                            lats[iy], &zt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                            lats[iy], &tt[iy * nx + ix], ci, cw, 0);
        intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                            lats[iy], &qt[iy * nx + ix], ci, cw, 0);
      }

    /* Write data... */
    NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
    NC(nc_put_vara_double(ncid, clppid, start, count, pt));
    NC(nc_put_vara_double(ncid, clptid, start, count, tt));
    if (h2o)
      NC(nc_put_vara_double(ncid, clpqid, start, count, qt));

    /* Get dynamical tropopause... */
    ctl.met_tropo = 5;
    read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
    for (ix = 0; ix < nx; ix++)
      for (iy = 0; iy < ny; iy++) {
        intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                            &pt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                            lats[iy], &zt[iy * nx + ix], ci, cw, 1);
        intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
```

```
                          lats[iy], &tt[iy * nx + ix], ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                          lats[iy], &qt[iy * nx + ix], ci, cw, 0);
        }

      /* Write data... */
      NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
      NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
      NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
      if (h2o)
        NC(nc_put_vara_double(ncid, dynqid, start, count, qt));

      /* Get WMO 1st tropopause... */
      ctl.met_tropo = 3;
      read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
      for (ix = 0; ix < nx; ix++)
        for (iy = 0; iy < ny; iy++) {
          intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                          &pt[iy * nx + ix], ci, cw, 1);
          intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                          lats[iy], &zt[iy * nx + ix], ci, cw, 1);
          intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                          lats[iy], &tt[iy * nx + ix], ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                          lats[iy], &qt[iy * nx + ix], ci, cw, 0);
        }

      /* Write data... */
      NC(nc_put_vara_double(ncid, wmo1zid, start, count, zt));
      NC(nc_put_vara_double(ncid, wmo1pid, start, count, pt));
      NC(nc_put_vara_double(ncid, wmo1tid, start, count, tt));
      if (h2o)
        NC(nc_put_vara_double(ncid, wmo1qid, start, count, qt));

      /* Get WMO 2nd tropopause... */
      ctl.met_tropo = 4;
      read_met_tropo(&ctl, met);
#pragma omp parallel for default(shared) private(ix,iy,ci,cw)
      for (ix = 0; ix < nx; ix++)
        for (iy = 0; iy < ny; iy++) {
          intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
                          &pt[iy * nx + ix], ci, cw, 1);
          intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
                          lats[iy], &zt[iy * nx + ix], ci, cw, 1);
          intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
                          lats[iy], &tt[iy * nx + ix], ci, cw, 0);
          intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
                          lats[iy], &qt[iy * nx + ix], ci, cw, 0);
        }

      /* Write data... */
      NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
      NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
      NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
      if (h2o)
        NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));

      /* Increment time step counter... */
      nt++;
    }

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

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

  return EXIT_SUCCESS;
}

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

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

  int varid;

  NC(nc_inq_varid(ncid, varname, &varid));
  NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
}
```

5.37 tropo_sample.c File Reference

Sample tropopause climatology.

Functions

- double intpol_help (double x0, double y0, double x1, double y1, double x)
- double intpol_2d (float array[EX][EY], double lons[EX], double lats[EY], size_t nlon, size_t nlat, double lon, double lat)
- int main (int argc, char ∗argv[ ])

5.37.1 Detailed Description

Sample tropopause climatology.

Definition in file tropo_sample.c.

5.37.2 Function Documentation

5.37.2.1 double intpol_help (double *x0*, double *y0*, double *x1*, double *y1*, double *x*)

Definition at line 269 of file tropo_sample.c.

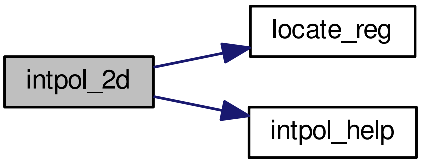
```
            {

  /* Linear interpolation... */
  if (gsl_finite(y0) && gsl_finite(y1))
    return LIN(x0, y0, x1, y1, x);

  /* Nearest neighbour... */
  else {
    if (fabs(x - x0) < fabs(x - x1))
      return y0;
    else
      return y1;
  }
}
```

5.37.2.2 double intpol_2d (float *array[EX][EY]*, double *lons[EX]*, double *lats[EY]*, size_t *nlon*, size_t *nlat*, double *lon*, double *lat*)

Definition at line 291 of file tropo_sample.c.

```
              {

  double aux0, aux1;

  /* Adjust longitude... */
  if (lon < lons[0])
    lon += 360;
  else if (lon > lons[nlon - 1])
    lon -= 360;

  /* Get indices... */
  int ix = locate_reg(lons, (int) nlon, lon);
  int iy = locate_reg(lats, (int) nlat, lat);

  /* Interpolate in longitude... */
  aux0 = intpol_help(lons[ix], array[ix][iy],
                     lons[ix + 1], array[ix + 1][iy], lon);
  aux1 = intpol_help(lons[ix], array[ix][iy + 1],
                     lons[ix + 1], array[ix + 1][iy + 1], lon);

  /* Interpolate in latitude... */
  return intpol_help(lats[iy], aux0, lats[iy + 1], aux1, lat);
}
```

Here is the call graph for this function:



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

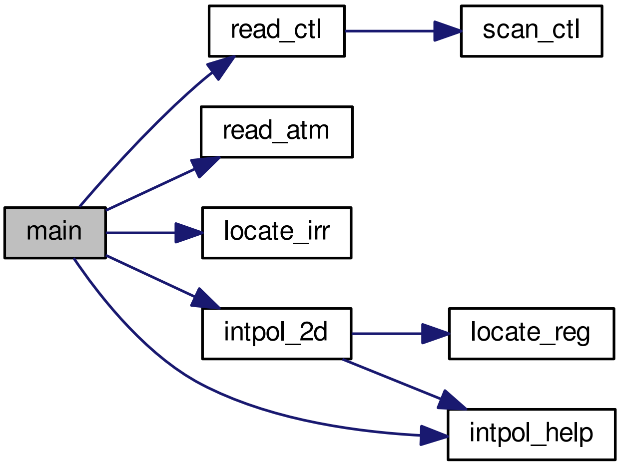
Definition at line 60 of file tropo_sample.c.

```
                  {

  ctl_t ctl;

  atm_t *atm;

  static FILE *out;

  static char varname[LEN];

  static double times[NT], lons[EX], lats[EY], time0, time1, z0, z1, p0, p1,
    t0, t1, q0, q1;

  static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
    tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
    tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];

  static int ip, iq, it, it_old = -999, dimid[10], ncid,
    varid, varid_z, varid_p, varid_t, varid_q, h2o;

  static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;

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

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");

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

  /* Read atmospheric data... */
  if (!read_atm(argv[5], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Open tropopause file... */
  printf("Read tropopause data: %s\n", argv[3]);
  if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
    ERRMSG("Cannot open file!");

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "time", &dimid[0]));
  NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
  if (ntime > NT)
    ERRMSG("Too many times!");
  NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
  NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
  if (nlat > EY)
    ERRMSG("Too many latitudes!");
  NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
  NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
  if (nlon > EX)
    ERRMSG("Too many longitudes!");

```

```
/* Read coordinates... */
NC(nc_inq_varid(ncid, "time", &varid));
NC(nc_get_var_double(ncid, varid, times));
NC(nc_inq_varid(ncid, "lat", &varid));
NC(nc_get_var_double(ncid, varid, lats));
NC(nc_inq_varid(ncid, "lon", &varid));
NC(nc_get_var_double(ncid, varid, lons));

/* Get variable indices... */
sprintf(varname, "%s_z", argv[4]);
NC(nc_inq_varid(ncid, varname, &varid_z));
sprintf(varname, "%s_p", argv[4]);
NC(nc_inq_varid(ncid, varname, &varid_p));
sprintf(varname, "%s_t", argv[4]);
NC(nc_inq_varid(ncid, varname, &varid_t));
sprintf(varname, "%s_q", argv[4]);
h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);

/* Set dimensions... */
count[0] = 1;
count[1] = nlat;
count[2] = nlon;

/* Create file... */
printf("Write tropopause sample data: %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");
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, "# $%d = tropopause height [km]\n", 5 + ctl.nq);
fprintf(out, "# $%d = tropopause pressure [hPa]\n", 6 + ctl.nq);
fprintf(out, "# $%d = tropopause temperature [K]\n", 7 + ctl.nq);
fprintf(out, "# $%d = tropopause water vapor [ppv]\n\n", 8 + ctl.nq);

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

  /* Check temporal ordering... */
  if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
    ERRMSG("Time must be ascending!");

  /* Check range... */
  if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
    continue;

  /* Read data... */
  it = locate_irr(times, (int) ntime, atm->time[ip]);
  if (it != it_old) {

    time0 = times[it];
    start[0] = (size_t) it;
    NC(nc_get_vara_float(ncid, varid_z, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_p, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_t, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
    if (h2o) {
      NC(nc_get_vara_float(ncid, varid_q, start, count, help));
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
    } else
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_q0[ilon][ilat] = GSL_NAN;

    time1 = times[it + 1];
    start[0] = (size_t) it + 1;
    NC(nc_get_vara_float(ncid, varid_z, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_p, start, count, help));
```

```
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
      NC(nc_get_vara_float(ncid, varid_t, start, count, help));
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
      if (h2o) {
        NC(nc_get_vara_float(ncid, varid_q, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_q1[ilon][ilat] = GSL_NAN;;
    }
    it_old = it;

    /* Interpolate... */
    z0 = intpol_2d(tropo_z0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    p0 = intpol_2d(tropo_p0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    t0 = intpol_2d(tropo_t0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    q0 = intpol_2d(tropo_q0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);

    z1 = intpol_2d(tropo_z1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    p1 = intpol_2d(tropo_p1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    t1 = intpol_2d(tropo_t1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    q1 = intpol_2d(tropo_q1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);

    z0 = intpol_help(time0, z0, time1, z1, atm->time[ip]);
    p0 = intpol_help(time0, p0, time1, p1, atm->time[ip]);
    t0 = intpol_help(time0, t0, time1, t1, atm->time[ip]);
    q0 = intpol_help(time0, q0, time1, q1, atm->time[ip]);

    /* 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, " %g %g %g %g\n", z0, p0, t0, q0);
  }

  /* Close files... */
  fclose(out);
  NC(nc_close(ncid));

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.38 tropo_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"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define NT 744

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Linear interpolation considering missing values. */
double intpol_help(
  double x0,
  double y0,
  double x1,
  double y1,
  double x);

/* Bilinear horizontal interpolation. */
double intpol_2d(
  float array[EX][EY],
  double lons[EX],
  double lats[EY],
```

```
  size_t nlon,
  size_t nlat,
  double lon,
  double lat);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

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

  ctl_t ctl;

  atm_t *atm;

  static FILE *out;

  static char varname[LEN];

  static double times[NT], lons[EX], lats[EY], time0, time1, z0, z1, p0, p1,
    t0, t1, q0, q1;

  static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
    tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
    tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];

  static int ip, iq, it, it_old = -999, dimid[10], ncid,
    varid, varid_z, varid_p, varid_t, varid_q, h2o;

  static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;

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

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");

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

  /* Read atmospheric data... */
  if (!read_atm(argv[5], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Open tropopause file... */
  printf("Read tropopause data: %s\n", argv[3]);
  if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
    ERRMSG("Cannot open file!");

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "time", &dimid[0]));
  NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
  if (ntime > NT)
    ERRMSG("Too many times!");
  NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
  NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
  if (nlat > EY)
    ERRMSG("Too many latitudes!");
  NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
  NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
  if (nlon > EX)
    ERRMSG("Too many longitudes!");

  /* Read coordinates... */
  NC(nc_inq_varid(ncid, "time", &varid));
  NC(nc_get_var_double(ncid, varid, times));
  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, lats));
  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, lons));

  /* Get variable indices... */
  sprintf(varname, "%s_z", argv[4]);
  NC(nc_inq_varid(ncid, varname, &varid_z));
  sprintf(varname, "%s_p", argv[4]);
  NC(nc_inq_varid(ncid, varname, &varid_p));
  sprintf(varname, "%s_t", argv[4]);
  NC(nc_inq_varid(ncid, varname, &varid_t));
  sprintf(varname, "%s_q", argv[4]);
  h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);

  /* Set dimensions... */
  count[0] = 1;
  count[1] = nlat;
```

```
count[2] = nlon;

/* Create file... */
printf("Write tropopause sample data: %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");
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, "# $%d = tropopause height [km]\n", 5 + ctl.nq);
fprintf(out, "# $%d = tropopause pressure [hPa]\n", 6 + ctl.nq);
fprintf(out, "# $%d = tropopause temperature [K]\n", 7 + ctl.nq);
fprintf(out, "# $%d = tropopause water vapor [ppv]\n\n", 8 + ctl.nq);

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

  /* Check temporal ordering... */
  if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
    ERRMSG("Time must be ascending!");

  /* Check range... */
  if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
    continue;

  /* Read data... */
  it = locate_irr(times, (int) ntime, atm->time[ip]);
  if (it != it_old) {

    time0 = times[it];
    start[0] = (size_t) it;
    NC(nc_get_vara_float(ncid, varid_z, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_p, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_t, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
    if (h2o) {
      NC(nc_get_vara_float(ncid, varid_q, start, count, help));
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
    } else
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_q0[ilon][ilat] = GSL_NAN;

    time1 = times[it + 1];
    start[0] = (size_t) it + 1;
    NC(nc_get_vara_float(ncid, varid_z, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_p, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
    NC(nc_get_vara_float(ncid, varid_t, start, count, help));
    for (ilon = 0; ilon < nlon; ilon++)
      for (ilat = 0; ilat < nlat; ilat++)
        tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
    if (h2o) {
      NC(nc_get_vara_float(ncid, varid_q, start, count, help));
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
    } else
      for (ilon = 0; ilon < nlon; ilon++)
        for (ilat = 0; ilat < nlat; ilat++)
          tropo_q1[ilon][ilat] = GSL_NAN;;
  }
  it_old = it;

  /* Interpolate... */
  z0 = intpol_2d(tropo_z0, lons, lats, nlon, nlat,
```

```
                   atm->lon[ip], atm->lat[ip]);
    p0 = intpol_2d(tropo_p0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    t0 = intpol_2d(tropo_t0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    q0 = intpol_2d(tropo_q0, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);

    z1 = intpol_2d(tropo_z1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    p1 = intpol_2d(tropo_p1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    t1 = intpol_2d(tropo_t1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);
    q1 = intpol_2d(tropo_q1, lons, lats, nlon, nlat,
                   atm->lon[ip], atm->lat[ip]);

    z0 = intpol_help(time0, z0, time1, z1, atm->time[ip]);
    p0 = intpol_help(time0, p0, time1, p1, atm->time[ip]);
    t0 = intpol_help(time0, t0, time1, t1, atm->time[ip]);
    q0 = intpol_help(time0, q0, time1, q1, atm->time[ip]);

    /* 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, " %g %g %g %g\n", z0, p0, t0, q0);
  }

  /* Close files... */
  fclose(out);
  NC(nc_close(ncid));

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

  return EXIT_SUCCESS;
}

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

double intpol_help(
  double x0,
  double y0,
  double x1,
  double y1,
  double x) {

  /* Linear interpolation... */
  if (gsl_finite(y0) && gsl_finite(y1))
    return LIN(x0, y0, x1, y1, x);

  /* Nearest neighbour... */
  else {
    if (fabs(x - x0) < fabs(x - x1))
      return y0;
    else
      return y1;
  }
}

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

double intpol_2d(
  float array[EX][EY],
  double lons[EX],
  double lats[EY],
  size_t nlon,
  size_t nlat,
  double lon,
  double lat) {

  double aux0, aux1;

  /* Adjust longitude... */
  if (lon < lons[0])
    lon += 360;
  else if (lon > lons[nlon - 1])
    lon -= 360;

  /* Get indices... */
  int ix = locate_reg(lons, (int) nlon, lon);
  int iy = locate_reg(lats, (int) nlat, lat);

```

```
  /* Interpolate in longitude... */
  aux0 = intpol_help(lons[ix], array[ix][iy],
                     lons[ix + 1], array[ix + 1][iy], lon);
  aux1 = intpol_help(lons[ix], array[ix][iy + 1],
                     lons[ix + 1], array[ix + 1][iy + 1], lon);

  /* Interpolate in latitude... */
  return intpol_help(lats[iy], aux0, lats[iy + 1], aux1, lat);
}
```

Index