MPTRAC

Generated by Doxygen 1.9.1

1 Main Page

Massive-Parallel Trajectory Calculations (MPTRAC) is a Lagrangian particle dispersion model for the free 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 parcels.
- double time [NP]

  Time [s].
- double p [NP]

  Pressure [hPa].
- double zeta [NP]

  Zeta [K].

- 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 1373 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 np `int atm_t::np`

Number of air parcels.

Definition at line 1376 of file libtrac.h.

4.1.2.2 time `double atm_t::time[NP]`

Time [s].

Definition at line 1379 of file libtrac.h.

4.1.2.3 p `double atm_t::p[NP]`

Pressure [hPa].

Definition at line 1382 of file libtrac.h.

4.1.2.4 zeta `double atm_t::zeta[NP]`

Zeta [K].

Definition at line 1385 of file libtrac.h.

4.1.2.5 lon `double atm_t::lon[NP]`

Longitude [deg].

Definition at line 1388 of file libtrac.h.

4.1.2.6 lat `double atm_t::lat[NP]`

Latitude [deg].

Definition at line 1391 of file libtrac.h.

4.1.2.7 q `double atm_t::q[NQ][NP]`

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

Definition at line 1394 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

- 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.
- float uvwp [NP][3]

  Wind perturbations [m/s].

4.2.1 Detailed Description

Cache data.

Definition at line 1399 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 iso_var `double cache_t::iso_var[NP]`

Isosurface variables.

Definition at line 1402 of file libtrac.h.

4.2.2.2 iso_ps `double cache_t::iso_ps[NP]`

Isosurface balloon pressure [hPa].

Definition at line 1405 of file libtrac.h.

4.2.2.3 iso_ts `double cache_t::iso_ts[NP]`

Isosurface balloon time [s].

Definition at line 1408 of file libtrac.h.

4.2.2.4 iso_n `int cache_t::iso_n`

Isosurface balloon number of data points.

Definition at line 1411 of file libtrac.h.

4.2.2.5 uvwp `float cache_t::uvwp[NP][3]`

Wind perturbations [m/s].

Definition at line 1414 of file libtrac.h.

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

- libtrac.h

4.3 clim_t Struct Reference

Climatological data.

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

Data Fields

- int tropo_ntime
  Number of tropopause timesteps.
- int tropo_nlat
  Number of tropopause latitudes.
- double tropo_time [12]
  Tropopause time steps [s].
- double tropo_lat [73]
  Tropopause latitudes [deg].
- double tropo [12][73]
  Tropopause pressure values [hPa].
- int hno3_ntime
  Number of HNO3 timesteps.
- int hno3_nlat
  Number of HNO3 latitudes.
- int hno3_np
  Number of HNO3 pressure levels.
- double hno3_time [12]
  HNO3 time steps [s].
- double hno3_lat [18]
  HNO3 latitudes [deg].
- double hno3_p [10]
  HNO3 pressure levels [hPa].
- double hno3 [12][18][10]
  HNO3 volume mixing ratios [ppv].
- int oh_ntime
  Number of OH timesteps.
- int oh_nlat
  Number of OH latitudes.
- int oh_np
  Number of OH pressure levels.
- double oh_time [CT]
  OH time steps [s].
- double oh_lat [CY]
  OH latitudes [deg].
- double oh_p [CP]
  OH pressure levels [hPa].
- double oh [CT][CP][CY]
  OH number concentrations [molec/cm^3].
- int h2o2_ntime
  Number of H2O2 timesteps.
- int h2o2_nlat
  Number of H2O2 latitudes.
- int h2o2_np
  Number of H2O2 pressure levels.
- double h2o2_time [CT]
  H2O2 time steps [s].
- double h2o2_lat [CY]
  H2O2 latitudes [deg].
- double h2o2_p [CP]
  H2O2 pressure levels [hPa].
- double h2o2 [CT][CP][CY]
  H2O2 number concentrations [molec/cm^3].

4.3.1 Detailed Description

Climatological data.

Definition at line 1419 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 tropo_ntime `int clim_t::tropo_ntime`

Number of tropopause timesteps.

Definition at line 1422 of file libtrac.h.

4.3.2.2 tropo_nlat `int clim_t::tropo_nlat`

Number of tropopause latitudes.

Definition at line 1425 of file libtrac.h.

4.3.2.3 tropo_time `double clim_t::tropo_time[12]`

Tropopause time steps [s].

Definition at line 1428 of file libtrac.h.

4.3.2.4 tropo_lat `double clim_t::tropo_lat[73]`

Tropopause latitudes [deg].

Definition at line 1431 of file libtrac.h.

4.3.2.5 tropo `double clim_t::tropo[12][73]`

Tropopause pressure values [hPa].

Definition at line 1434 of file libtrac.h.

4.3.2.6 hno3_ntime `int clim_t::hno3_ntime`

Number of HNO3 timesteps.

Definition at line 1437 of file libtrac.h.

4.3.2.7 hno3_nlat `int clim_t::hno3_nlat`

Number of HNO3 latitudes.

Definition at line 1440 of file libtrac.h.

4.3.2.8 hno3_np `int clim_t::hno3_np`

Number of HNO3 pressure levels.

Definition at line 1443 of file libtrac.h.

4.3.2.9 hno3_time `double clim_t::hno3_time[12]`

HNO3 time steps [s].

Definition at line 1446 of file libtrac.h.

4.3.2.10 hno3_lat `double clim_t::hno3_lat[18]`

HNO3 latitudes [deg].

Definition at line 1449 of file libtrac.h.

4.3.2.11 hno3_p `double clim_t::hno3_p[10]`

HNO3 pressure levels [hPa].

Definition at line 1452 of file libtrac.h.

4.3.2.12 hno3 `double clim_t::hno3[12][18][10]`

HNO3 volume mixing ratios [ppv].

Definition at line 1455 of file libtrac.h.

4.3.2.13 oh_ntime `int clim_t::oh_ntime`

Number of OH timesteps.

Definition at line 1458 of file libtrac.h.

4.3.2.14 oh_nlat `int clim_t::oh_nlat`

Number of OH latitudes.

Definition at line 1461 of file libtrac.h.

4.3.2.15 oh_np `int clim_t::oh_np`

Number of OH pressure levels.

Definition at line 1464 of file libtrac.h.

4.3.2.16 oh_time `double clim_t::oh_time[CT]`

OH time steps [s].

Definition at line 1467 of file libtrac.h.

4.3.2.17 oh_lat `double clim_t::oh_lat[CY]`

OH latitudes [deg].

Definition at line 1470 of file libtrac.h.

4.3.2.18 oh_p `double clim_t::oh_p[CP]`

OH pressure levels [hPa].

Definition at line 1473 of file libtrac.h.

4.3.2.19 oh `double clim_t::oh[CT][CP][CY]`

OH number concentrations [molec/cm^3].

Definition at line 1476 of file libtrac.h.

4.3.2.20 h2o2_ntime `int clim_t::h2o2_ntime`

Number of H2O2 timesteps.

Definition at line 1479 of file libtrac.h.

4.3.2.21 h2o2_nlat `int clim_t::h2o2_nlat`

Number of H2O2 latitudes.

Definition at line 1482 of file libtrac.h.

4.3.2.22 h2o2_np `int clim_t::h2o2_np`

Number of H2O2 pressure levels.

Definition at line 1485 of file libtrac.h.

4.3.2.23 h2o2_time `double clim_t::h2o2_time[CT]`

H2O2 time steps [s].

Definition at line 1488 of file libtrac.h.

4.3.2.24 h2o2_lat `double clim_t::h2o2_lat[CY]`

H2O2 latitudes [deg].

Definition at line 1491 of file libtrac.h.

4.3.2.25 h2o2_p `double clim_t::h2o2_p[CP]`

H2O2 pressure levels [hPa].

Definition at line 1494 of file libtrac.h.

4.3.2.26 h2o2 `double clim_t::h2o2[CT][CP][CY]`

H2O2 number concentrations [molec/cm^3].

Definition at line 1497 of file libtrac.h.

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

- libtrac.h

4.4 ctl_t Struct Reference

Control parameters.

`#include <libtrac.h>`

Data Fields

- int vert_coord_ap

  Vertical coordinate of air parcels (0=pressure, 1=zeta).
- int vert_coord_met

  Vertical coordinate of input meteo data (0=automatic, 1=eta).
- int vert_vel

  Vertical velocity (0=kinematic, 1=diabatic).
- int clams_met_data

  Read MPTRAC or CLaMS meteo data (0=MPTRAC, 1=CLaMS).
- size_t chunkszhint

  Chunk size hint for nc__open.
- int read_mode

  Read mode for nc__open.
- int nq

  Number of quantities.
- char qnt_name [NQ][LEN]

  Quantity names.

- char qnt_longname [NQ][LEN]

  Quantity long names.
- char qnt_unit [NQ][LEN]

  Quantity units.
- char qnt_format [NQ][LEN]

  Quantity output format.
- int qnt_idx

  Quantity array index for air parcel IDs.
- int qnt_ens

  Quantity array index for ensemble IDs.
- int qnt_stat

  Quantity array index for station flag.
- int qnt_m

  Quantity array index for mass.
- int qnt_vmr

  Quantity array index for volume mixing ratio.
- int qnt_rp

  Quantity array index for particle radius.
- int qnt_rhop

  Quantity array index for particle density.
- int qnt_ps

  Quantity array index for surface pressure.
- int qnt_ts

  Quantity array index for surface temperature.
- int qnt_zs

  Quantity array index for surface geopotential height.
- int qnt_us

  Quantity array index for surface zonal wind.
- int qnt_vs

  Quantity array index for surface meridional wind.
- int qnt_pbl

  Quantity array index for boundary layer pressure.
- int qnt_pt

  Quantity array index for tropopause pressure.
- int qnt_tt

  Quantity array index for tropopause temperature.
- int qnt_zt

  Quantity array index for tropopause geopotential height.
- int qnt_h2ot

  Quantity array index for tropopause water vapor vmr.
- 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_rho

  Quantity array index for density of air.
- 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_pct

  Quantity array index for cloud top pressure.
- int qnt_pcb

  Quantity array index for cloud bottom pressure.
- int qnt_cl

  Quantity array index for total column cloud water.
- int qnt_plcl

  Quantity array index for pressure at lifted condensation level (LCL).
- int qnt_plfc

  Quantity array index for pressure at level of free convection (LCF).
- int qnt_pel

  Quantity array index for pressure at equilibrium level (EL).
- int qnt_cape

  Quantity array index for convective available potential energy (CAPE).
- int qnt_cin

  Quantity array index for convective inhibition (CIN).
- int qnt_hno3

  Quantity array index for nitric acid vmr.
- int qnt_oh

  Quantity array index for hydroxyl number concentrations.
- int qnt_vmrimpl

  Quantity array index for implicity volumn mixing ratio.
- int qnt_mloss_oh

  Quantity array index for total mass loss due to OH chemistry.
- int qnt_mloss_h2o2

  Quantity array index for total mass loss due to H2O2 chemistry.
- int qnt_mloss_wet

  Quantity array index for total mass loss due to wet deposition.
- int qnt_mloss_dry

  Quantity array index for total mass loss due to dry deposition.
- int qnt_mloss_decay

  Quantity array index for total mass loss due to exponential decax.
- int qnt_psat

  Quantity array index for saturation pressure over water.
- int qnt_psice

  Quantity array index for saturation pressure over ice.
- int qnt_pw

  Quantity array index for partial water vapor pressure.
- int qnt_sh

  Quantity array index for specific humidity.

- int qnt_rh

  Quantity array index for relative humidity over water.
- int qnt_rhice

  Quantity array index for relative humidity over ice.
- int qnt_theta

  Quantity array index for potential temperature.
- int qnt_zeta

  Quantity array index for zeta vertical coordinate.
- int qnt_tvirt

  Quantity array index for virtual temperature.
- int qnt_lapse

  Quantity array index for lapse rate.
- 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_tdew

  Quantity array index for dew point temperature.
- 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 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].
- char metbase [LEN]

  Basename for meteo data.
- double dt_met

  Time step of meteo data [s].
- int met_type

  Type of meteo data files (0=netCDF, 1=binary, 2=pack, 3=zfp, 4=zstd).
- int met_nc_scale

  Check netCDF scaling factors (0=no, 1=yes).
- 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.
- double met_detrend

  FWHM of horizontal Gaussian used for detrending [km].
- int met_np

  Number of target pressure levels.
- double met_p [EP]

  Target pressure levels [hPa].
- int met_geopot_sx

  Longitudinal smoothing of geopotential heights.
- int met_geopot_sy

  Latitudinal smoothing of geopotential heights.
- int met_relhum

  Try to read relative humidity (0=no, 1=yes).
- int met_tropo

  Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO_1st, 4=WMO_2nd, 5=dynamical).
- double met_tropo_lapse

  WMO tropopause lapse rate [K/km].
- int met_tropo_nlev

  WMO tropopause layer depth (number of levels).
- double met_tropo_lapse_sep

  WMO tropopause separation layer lapse rate [K/km].
- int met_tropo_nlev_sep

  WMO tropopause separation layer depth (number of levels).
- double met_tropo_pv

  Dyanmical tropopause potential vorticity threshold [PVU].
- double met_tropo_theta

  Dynamical tropopause potential temperature threshold [K].
- int met_tropo_spline

  Tropopause interpolation method (0=linear, 1=spline).
- int met_cloud

  Cloud data (0=none, 1=LWC+IWC, 2=RWC+SWC, 3=all).
- double met_cloud_min

  Minimum cloud ice water content [kg/kg].
- double met_dt_out

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

  Preload meteo data into disk cache (0=no, 1=yes).
- double sort_dt

  Time step for sorting of particle data [s].
- int isosurf

  Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
- char balloon [LEN]

  Balloon position filename.
- int advect

  Advection scheme (1=Euler, 2=midpoint, 4=Runge-Kutta).
- int reflect

  Reflection of particles at top and bottom boundary (0=no, 1=yes).
- double turb_dx_trop

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

- double turb_dx_strat

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

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

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

  Horizontal scaling factor for mesoscale wind fluctuations.
- double turb_mesoz

  Vertical scaling factor for mesoscale wind fluctuations.
- double conv_cape

  CAPE threshold for convection module [J/kg].
- double conv_cin

  CIN threshold for convection module [J/kg].
- double conv_dt

  Time interval for convection module [s].
- int conv_mix

  Type of vertical mixing (0=pressure, 1=density).
- int conv_mix_bot

  Lower level for mixing (0=particle pressure, 1=surface).
- int conv_mix_top

  Upper level for mixing (0=particle pressure, 1=EL).
- double bound_mass

  Boundary conditions mass per particle [kg].
- double bound_mass_trend

  Boundary conditions mass per particle trend [kg/s].
- double bound_vmr

  Boundary conditions volume mixing ratio [ppv].
- double bound_vmr_trend

  Boundary conditions volume mixing ratio trend [ppv/s].
- double bound_lat0

  Boundary conditions minimum longitude [deg].
- double bound_lat1

  Boundary conditions maximum longitude [deg].
- double bound_p0

  Boundary conditions bottom pressure [hPa].
- double bound_p1

  Boundary conditions top pressure [hPa].
- double bound_dps

  Boundary conditions surface layer depth [hPa].
- double bound_dzs

  Boundary conditions surface layer depth [km].
- double bound_zetas

  Boundary conditions surface layer zeta [K].
- int bound_pbl

  Boundary conditions planetary boundary layer (0=no, 1=yes).
- 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].
- char clim_oh_filename [LEN]

 Filename of OH climatology.
- char clim_h2o2_filename [LEN]

 Filename of H2O2 climatology.
- int oh_chem_reaction

 Reaction type for OH chemistry (0=none, 2=bimolecular, 3=termolecular).
- double oh_chem [4]

 Coefficients for OH reaction rate (A, E/R or k0, n, kinf, m).
- double oh_chem_beta

 Beta parameter for diurnal variablity of OH.
- double h2o2_chem_cc

 Cloud cover parameter for H2O2 chemistry.
- int h2o2_chem_reaction

 Reaction type for H2O2 chemistry (0=none, 1=SO2).
- int chemgrid_nz

 Number of altitudes of chemistry grid.
- double chemgrid_z0

 Lower altitude of chemistry grid [km].
- double chemgrid_z1

 Upper altitude of chemistry grid [km].
- int chemgrid_nx

 Number of longitudes of chemistry grid.
- double chemgrid_lon0

 Lower longitude of chemistry grid [deg].
- double chemgrid_lon1

 Upper longitude of chemistry grid [deg].
- int chemgrid_ny

 Number of latitudes of chemistry grid.
- double chemgrid_lat0

 Lower latitude of chemistry grid [deg].
- double chemgrid_lat1

 Upper latitude of chemistry grid [deg].
- double dry_depo_dp

 Dry deposition surface layer [hPa].
- double dry_depo_vdep

 Dry deposition velocity [m/s].
- double wet_depo_pre [2]

 Coefficients for precipitation calculation.
- double wet_depo_bc_a

 Coefficient A for wet deposition below cloud (exponential form).
- double wet_depo_bc_b

 Coefficient B for wet deposition below cloud (exponential form).
- double wet_depo_ic_a

 Coefficient A for wet deposition in cloud (exponential form).
- double wet_depo_ic_b

 Coefficient B for wet deposition in cloud (exponential form).
- double wet_depo_ic_h [3]

 Coefficients for wet deposition in cloud (Henry's law: Hb, Cb, pH).

- double wet_depo_bc_h [2]

  Coefficients for wet deposition below cloud (Henry's law: Hb, Cb).
- double wet_depo_ic_ret_ratio

  Coefficients for wet deposition in cloud: retention ratio.
- double wet_depo_bc_ret_ratio

  Coefficients for wet deposition below cloud: retention ratio.
- 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=none, 1=missval, 2=remove).
- int atm_stride

  Particle index stride for atmospheric data files.
- int atm_type

  Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF, 3=CLaMS).
- 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 ens_basename [LEN]

Basename of ensemble data file.
- double ens_dt_out

    Time step for ensemble output [s].
- 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].
- int grid_type

    Type of grid data files (0=ASCII, 1=netCDF).
- 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 sample_basename [LEN]

  Basename of sample data file.
- char sample_obsfile [LEN]

  Observation data file for sample output.
- double sample_dx

  Horizontal radius for sample output [km].
- double sample_dz

  Layer depth for sample output [km].
- 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].
- double stat_t0

  Start time for station output [s].
- double stat_t1

  Stop time for station output [s].

4.4.1 Detailed Description

Control parameters.

Definition at line 697 of file libtrac.h.

4.4.2 Field Documentation

4.4.2.1 vert_coord_ap `int ctl_t::vert_coord_ap`

Vertical coordinate of air parcels (0=pressure, 1=zeta).

Definition at line 700 of file libtrac.h.

4.4.2.2 vert_coord_met `int ctl_t::vert_coord_met`

Vertical coordinate of input meteo data (0=automatic, 1=eta).

Definition at line 703 of file libtrac.h.

4.4.2.3 vert_vel `int ctl_t::vert_vel`

Vertical velocity (0=kinematic, 1=diabatic).

Definition at line 706 of file libtrac.h.

4.4.2.4 clams_met_data `int ctl_t::clams_met_data`

Read MPTRAC or CLaMS meteo data (0=MPTRAC, 1=CLaMS).

Definition at line 709 of file libtrac.h.

4.4.2.5 chunkszhint `size_t ctl_t::chunkszhint`

Chunk size hint for nc__open.

Definition at line 712 of file libtrac.h.

4.4.2.6 read_mode `int ctl_t::read_mode`

Read mode for nc__open.

Definition at line 715 of file libtrac.h.

4.4.2.7 nq `int ctl_t::nq`

Number of quantities.

Definition at line 718 of file libtrac.h.

4.4.2.8 qnt_name `char ctl_t::qnt_name[NQ][LEN]`

Quantity names.

Definition at line 721 of file libtrac.h.

4.4.2.9 qnt_longname `char ctl_t::qnt_longname[NQ][LEN]`

Quantity long names.

Definition at line 724 of file libtrac.h.

4.4.2.10 qnt_unit `char ctl_t::qnt_unit[NQ][LEN]`

Quantity units.

Definition at line 727 of file libtrac.h.

4.4.2.11 qnt_format `char ctl_t::qnt_format[NQ][LEN]`

Quantity output format.

Definition at line 730 of file libtrac.h.

4.4.2.12 qnt_idx `int ctl_t::qnt_idx`

Quantity array index for air parcel IDs.

Definition at line 733 of file libtrac.h.

4.4.2.13 qnt_ens `int ctl_t::qnt_ens`

Quantity array index for ensemble IDs.

Definition at line 736 of file libtrac.h.

4.4.2.14 qnt_stat `int ctl_t::qnt_stat`

Quantity array index for station flag.

Definition at line 739 of file libtrac.h.

4.4.2.15 qnt_m `int ctl_t::qnt_m`

Quantity array index for mass.

Definition at line 742 of file libtrac.h.

4.4.2.16 qnt_vmr `int ctl_t::qnt_vmr`

Quantity array index for volume mixing ratio.

Definition at line 745 of file libtrac.h.

4.4.2.17 qnt_rp `int ctl_t::qnt_rp`

Quantity array index for particle radius.

Definition at line 748 of file libtrac.h.

4.4.2.18 qnt_rhop `int ctl_t::qnt_rhop`

Quantity array index for particle density.

Definition at line 751 of file libtrac.h.

4.4.2.19 qnt_ps `int ctl_t::qnt_ps`

Quantity array index for surface pressure.

Definition at line 754 of file libtrac.h.

4.4.2.20 qnt_ts `int ctl_t::qnt_ts`

Quantity array index for surface temperature.

Definition at line 757 of file libtrac.h.

4.4.2.21 qnt_zs `int ctl_t::qnt_zs`

Quantity array index for surface geopotential height.

Definition at line 760 of file libtrac.h.

4.4.2.22 qnt_us `int ctl_t::qnt_us`

Quantity array index for surface zonal wind.

Definition at line 763 of file libtrac.h.

4.4.2.23 qnt_vs `int ctl_t::qnt_vs`

Quantity array index for surface meridional wind.

Definition at line 766 of file libtrac.h.

4.4.2.24 qnt_pbl `int ctl_t::qnt_pbl`

Quantity array index for boundary layer pressure.

Definition at line 769 of file libtrac.h.

4.4.2.25 qnt_pt `int ctl_t::qnt_pt`

Quantity array index for tropopause pressure.

Definition at line 772 of file libtrac.h.

4.4.2.26 qnt_tt `int ctl_t::qnt_tt`

Quantity array index for tropopause temperature.

Definition at line 775 of file libtrac.h.

4.4.2.27 qnt_zt `int ctl_t::qnt_zt`

Quantity array index for tropopause geopotential height.

Definition at line 778 of file libtrac.h.

4.4.2.28 qnt_h2ot `int ctl_t::qnt_h2ot`

Quantity array index for tropopause water vapor vmr.

Definition at line 781 of file libtrac.h.

4.4.2.29 qnt_z `int ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line 784 of file libtrac.h.

4.4.2.30 qnt_p `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line 787 of file libtrac.h.

4.4.2.31 qnt_t `int ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line 790 of file libtrac.h.

4.4.2.32 qnt_rho `int ctl_t::qnt_rho`

Quantity array index for density of air.

Definition at line 793 of file libtrac.h.

4.4.2.33 qnt_u `int ctl_t::qnt_u`

Quantity array index for zonal wind.

Definition at line 796 of file libtrac.h.

4.4.2.34 qnt_v `int ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line 799 of file libtrac.h.

4.4.2.35 qnt_w `int ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line 802 of file libtrac.h.

4.4.2.36 qnt_h2o `int ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line 805 of file libtrac.h.

4.4.2.37 qnt_o3 `int ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line 808 of file libtrac.h.

4.4.2.38 qnt_lwc `int ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line 811 of file libtrac.h.

4.4.2.39 qnt_iwc `int ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line 814 of file libtrac.h.

4.4.2.40 qnt_pct `int ctl_t::qnt_pct`

Quantity array index for cloud top pressure.

Definition at line 817 of file libtrac.h.

4.4.2.41 qnt_pcb `int ctl_t::qnt_pcb`

Quantity array index for cloud bottom pressure.

Definition at line 820 of file libtrac.h.

4.4.2.42 qnt_cl `int ctl_t::qnt_cl`

Quantity array index for total column cloud water.

Definition at line 823 of file libtrac.h.

4.4.2.43 qnt_plcl `int ctl_t::qnt_plcl`

Quantity array index for pressure at lifted condensation level (LCL).

Definition at line 826 of file libtrac.h.

4.4.2.44 qnt_plfc `int ctl_t::qnt_plfc`

Quantity array index for pressure at level of free convection (LCF).

Definition at line 829 of file libtrac.h.

4.4.2.45 qnt_pel `int ctl_t::qnt_pel`

Quantity array index for pressure at equilibrium level (EL).

Definition at line 832 of file libtrac.h.

4.4.2.46 qnt_cape `int ctl_t::qnt_cape`

Quantity array index for convective available potential energy (CAPE).

Definition at line 835 of file libtrac.h.

4.4.2.47 qnt_cin `int ctl_t::qnt_cin`

Quantity array index for convective inhibition (CIN).

Definition at line 838 of file libtrac.h.

4.4.2.48 qnt_hno3 `int ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line 841 of file libtrac.h.

4.4.2.49 qnt_oh `int ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line 844 of file libtrac.h.

4.4.2.50 qnt_vmrimpl `int ctl_t::qnt_vmrimpl`

Quantity array index for implicity volumn mixing ratio.

Definition at line 847 of file libtrac.h.

4.4.2.51 qnt_mloss_oh `int ctl_t::qnt_mloss_oh`

Quantity array index for total mass loss due to OH chemistry.

Definition at line 850 of file libtrac.h.

4.4.2.52 qnt_mloss_h2o2 `int ctl_t::qnt_mloss_h2o2`

Quantity array index for total mass loss due to H2O2 chemistry.

Definition at line 853 of file libtrac.h.

4.4.2.53 qnt_mloss_wet `int ctl_t::qnt_mloss_wet`

Quantity array index for total mass loss due to wet deposition.

Definition at line 856 of file libtrac.h.

4.4.2.54 qnt_mloss_dry `int ctl_t::qnt_mloss_dry`

Quantity array index for total mass loss due to dry deposition.

Definition at line 859 of file libtrac.h.

4.4.2.55 qnt_mloss_decay `int ctl_t::qnt_mloss_decay`

Quantity array index for total mass loss due to exponential decax.

Definition at line 862 of file libtrac.h.

4.4.2.56 qnt_psat `int ctl_t::qnt_psat`

Quantity array index for saturation pressure over water.

Definition at line 865 of file libtrac.h.

4.4.2.57 qnt_psice `int ctl_t::qnt_psice`

Quantity array index for saturation pressure over ice.

Definition at line 868 of file libtrac.h.

4.4.2.58 qnt_pw `int ctl_t::qnt_pw`

Quantity array index for partial water vapor pressure.

Definition at line 871 of file libtrac.h.

4.4.2.59 qnt_sh `int ctl_t::qnt_sh`

Quantity array index for specific humidity.

Definition at line 874 of file libtrac.h.

4.4.2.60 qnt_rh `int ctl_t::qnt_rh`

Quantity array index for relative humidity over water.

Definition at line 877 of file libtrac.h.

4.4.2.61 qnt_rhice `int ctl_t::qnt_rhice`

Quantity array index for relative humidity over ice.

Definition at line 880 of file libtrac.h.

4.4.2.62 qnt_theta `int ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line 883 of file libtrac.h.

4.4.2.63 qnt_zeta `int ctl_t::qnt_zeta`

Quantity array index for zeta vertical coordinate.

Definition at line 886 of file libtrac.h.

4.4.2.64 qnt_tvirt `int ctl_t::qnt_tvirt`

Quantity array index for virtual temperature.

Definition at line 889 of file libtrac.h.

4.4.2.65 qnt_lapse `int ctl_t::qnt_lapse`

Quantity array index for lapse rate.

Definition at line 892 of file libtrac.h.

4.4.2.66 qnt_vh `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line 895 of file libtrac.h.

4.4.2.67 qnt_vz `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line 898 of file libtrac.h.

4.4.2.68 qnt_pv `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line 901 of file libtrac.h.

4.4.2.69 qnt_tdew `int ctl_t::qnt_tdew`

Quantity array index for dew point temperature.

Definition at line 904 of file libtrac.h.

4.4.2.70 qnt_tice `int ctl_t::qnt_tice`

Quantity array index for T_ice.

Definition at line 907 of file libtrac.h.

4.4.2.71 qnt_tsts `int ctl_t::qnt_tsts`

Quantity array index for T_STS.

Definition at line 910 of file libtrac.h.

4.4.2.72 qnt_tnat `int ctl_t::qnt_tnat`

Quantity array index for T_NAT.

Definition at line 913 of file libtrac.h.

4.4.2.73 direction `int ctl_t::direction`

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

Definition at line 916 of file libtrac.h.

4.4.2.74 t_start `double ctl_t::t_start`

Start time of simulation [s].

Definition at line 919 of file libtrac.h.

4.4.2.75 t_stop `double ctl_t::t_stop`

Stop time of simulation [s].

Definition at line 922 of file libtrac.h.

4.4.2.76 dt_mod `double ctl_t::dt_mod`

Time step of simulation [s].

Definition at line 925 of file libtrac.h.

4.4.2.77 metbase `char ctl_t::metbase[LEN]`

Basename for meteo data.

Definition at line 928 of file libtrac.h.

4.4.2.78 dt_met `double ctl_t::dt_met`

Time step of meteo data [s].

Definition at line 931 of file libtrac.h.

4.4.2.79 met_type `int ctl_t::met_type`

Type of meteo data files (0=netCDF, 1=binary, 2=pack, 3=zfp, 4=zstd).

Definition at line 934 of file libtrac.h.

4.4.2.80 met_nc_scale `int ctl_t::met_nc_scale`

Check netCDF scaling factors (0=no, 1=yes).

Definition at line 937 of file libtrac.h.

4.4.2.81 met_dx `int ctl_t::met_dx`

Stride for longitudes.

Definition at line 940 of file libtrac.h.

4.4.2.82 met_dy `int ctl_t::met_dy`

Stride for latitudes.

Definition at line 943 of file libtrac.h.

4.4.2.83 met_dp `int ctl_t::met_dp`

Stride for pressure levels.

Definition at line 946 of file libtrac.h.

4.4.2.84 met_sx `int ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 949 of file libtrac.h.

4.4.2.85 met_sy `int ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 952 of file libtrac.h.

4.4.2.86 met_sp `int ctl_t::met_sp`

Smoothing for pressure levels.

Definition at line 955 of file libtrac.h.

4.4.2.87 met_detrend `double ctl_t::met_detrend`

FWHM of horizontal Gaussian used for detrending [km].

Definition at line 958 of file libtrac.h.

4.4.2.88 met_np `int ctl_t::met_np`

Number of target pressure levels.

Definition at line 961 of file libtrac.h.

4.4.2.89 met_p `double ctl_t::met_p[EP]`

Target pressure levels [hPa].

Definition at line 964 of file libtrac.h.

4.4.2.90 met_geopot_sx `int ctl_t::met_geopot_sx`

Longitudinal smoothing of geopotential heights.

Definition at line 967 of file libtrac.h.

4.4.2.91 met_geopot_sy `int ctl_t::met_geopot_sy`

Latitudinal smoothing of geopotential heights.

Definition at line 970 of file libtrac.h.

4.4.2.92 met_relhum `int ctl_t::met_relhum`

Try to read relative humidity (0=no, 1=yes).

Definition at line 973 of file libtrac.h.

4.4.2.93 met_tropo `int ctl_t::met_tropo`

Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO_1st, 4=WMO_2nd, 5=dynamical).

Definition at line 977 of file libtrac.h.

4.4.2.94 met_tropo_lapse `double ctl_t::met_tropo_lapse`

WMO tropopause lapse rate [K/km].

Definition at line 980 of file libtrac.h.

4.4.2.95 met_tropo_nlev `int ctl_t::met_tropo_nlev`

WMO tropopause layer depth (number of levels).

Definition at line 983 of file libtrac.h.

4.4.2.96 met_tropo_lapse_sep `double ctl_t::met_tropo_lapse_sep`

WMO tropopause separation layer lapse rate [K/km].

Definition at line 986 of file libtrac.h.

4.4.2.97 met_tropo_nlev_sep `int ctl_t::met_tropo_nlev_sep`

WMO tropopause separation layer depth (number of levels).

Definition at line 989 of file libtrac.h.

4.4.2.98 met_tropo_pv `double ctl_t::met_tropo_pv`

Dyanmical tropopause potential vorticity threshold [PVU].

Definition at line 992 of file libtrac.h.

4.4.2.99 met_tropo_theta `double ctl_t::met_tropo_theta`

Dynamical tropopause potential temperature threshold [K].

Definition at line 995 of file libtrac.h.

4.4.2.100 met_tropo_spline `int ctl_t::met_tropo_spline`

Tropopause interpolation method (0=linear, 1=spline).

Definition at line 998 of file libtrac.h.

4.4.2.101 met_cloud `int ctl_t::met_cloud`

Cloud data (0=none, 1=LWC+IWC, 2=RWC+SWC, 3=all).

Definition at line 1001 of file libtrac.h.

4.4.2.102 met_cloud_min `double ctl_t::met_cloud_min`

Minimum cloud ice water content [kg/kg].

Definition at line 1004 of file libtrac.h.

4.4.2.103 met_dt_out `double ctl_t::met_dt_out`

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

Definition at line 1007 of file libtrac.h.

4.4.2.104 met_cache `int ctl_t::met_cache`

Preload meteo data into disk cache (0=no, 1=yes).

Definition at line 1010 of file libtrac.h.

4.4.2.105 sort_dt `double ctl_t::sort_dt`

Time step for sorting of particle data [s].

Definition at line 1013 of file libtrac.h.

4.4.2.106 isosurf `int ctl_t::isosurf`

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

Definition at line 1017 of file libtrac.h.

4.4.2.107 balloon `char ctl_t::balloon[LEN]`

Balloon position filename.

Definition at line 1020 of file libtrac.h.

4.4.2.108 advect `int ctl_t::advect`

Advection scheme (1=Euler, 2=midpoint, 4=Runge-Kutta).

Definition at line 1023 of file libtrac.h.

4.4.2.109 reflect `int ctl_t::reflect`

Reflection of particles at top and bottom boundary (0=no, 1=yes).

Definition at line 1026 of file libtrac.h.

4.4.2.110 turb_dx_trop `double ctl_t::turb_dx_trop`

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

Definition at line 1029 of file libtrac.h.

4.4.2.111 turb_dx_strat `double ctl_t::turb_dx_strat`

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

Definition at line 1032 of file libtrac.h.

4.4.2.112 turb_dz_trop `double ctl_t::turb_dz_trop`

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

Definition at line 1035 of file libtrac.h.

4.4.2.113 turb_dz_strat `double ctl_t::turb_dz_strat`

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

Definition at line 1038 of file libtrac.h.

4.4.2.114 turb_mesox `double ctl_t::turb_mesox`

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 1041 of file libtrac.h.

4.4.2.115 turb_mesoz `double ctl_t::turb_mesoz`

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 1044 of file libtrac.h.

4.4.2.116 conv_cape `double ctl_t::conv_cape`

CAPE threshold for convection module [J/kg].

Definition at line 1047 of file libtrac.h.

4.4.2.117 conv_cin `double ctl_t::conv_cin`

CIN threshold for convection module [J/kg].

Definition at line 1050 of file libtrac.h.

4.4.2.118 conv_dt `double ctl_t::conv_dt`

Time interval for convection module [s].

Definition at line 1053 of file libtrac.h.

4.4.2.119 conv_mix `int ctl_t::conv_mix`

Type of vertical mixing (0=pressure, 1=density).

Definition at line 1056 of file libtrac.h.

4.4.2.120 conv_mix_bot `int ctl_t::conv_mix_bot`

Lower level for mixing (0=particle pressure, 1=surface).

Definition at line 1059 of file libtrac.h.

4.4.2.121 conv_mix_top `int ctl_t::conv_mix_top`

Upper level for mixing (0=particle pressure, 1=EL).

Definition at line 1062 of file libtrac.h.

4.4.2.122 bound_mass `double ctl_t::bound_mass`

Boundary conditions mass per particle [kg].

Definition at line 1065 of file libtrac.h.

4.4.2.123 bound_mass_trend `double ctl_t::bound_mass_trend`

Boundary conditions mass per particle trend [kg/s].

Definition at line 1068 of file libtrac.h.

4.4.2.124 bound_vmr `double ctl_t::bound_vmr`

Boundary conditions volume mixing ratio [ppv].

Definition at line 1071 of file libtrac.h.

4.4.2.125 bound_vmr_trend `double ctl_t::bound_vmr_trend`

Boundary conditions volume mixing ratio trend [ppv/s].

Definition at line 1074 of file libtrac.h.

4.4.2.126 bound_lat0 `double ctl_t::bound_lat0`

Boundary conditions minimum longitude [deg].

Definition at line 1077 of file libtrac.h.

4.4.2.127 bound_lat1 `double ctl_t::bound_lat1`

Boundary conditions maximum longitude [deg].

Definition at line 1080 of file libtrac.h.

4.4.2.128 bound_p0 `double ctl_t::bound_p0`

Boundary conditions bottom pressure [hPa].

Definition at line 1083 of file libtrac.h.

4.4.2.129 bound_p1 `double ctl_t::bound_p1`

Boundary conditions top pressure [hPa].

Definition at line 1086 of file libtrac.h.

4.4.2.130 bound_dps `double ctl_t::bound_dps`

Boundary conditions surface layer depth [hPa].

Definition at line 1089 of file libtrac.h.

4.4.2.131 bound_dzs `double ctl_t::bound_dzs`

Boundary conditions surface layer depth [km].

Definition at line 1092 of file libtrac.h.

4.4.2.132 bound_zetas `double ctl_t::bound_zetas`

Boundary conditions surface layer zeta [K].

Definition at line 1095 of file libtrac.h.

4.4.2.133 bound_pbl `int ctl_t::bound_pbl`

Boundary conditions planetary boundary layer (0=no, 1=yes).

Definition at line 1098 of file libtrac.h.

4.4.2.134 species `char ctl_t::species[LEN]`

Species.

Definition at line 1101 of file libtrac.h.

4.4.2.135 molmass `double ctl_t::molmass`

Molar mass [g/mol].

Definition at line 1104 of file libtrac.h.

4.4.2.136 tdec_trop `double ctl_t::tdec_trop`

Life time of particles (troposphere) [s].

Definition at line 1107 of file libtrac.h.

4.4.2.137 tdec_strat `double ctl_t::tdec_strat`

Life time of particles (stratosphere) [s].

Definition at line 1110 of file libtrac.h.

4.4.2.138 clim_oh_filename `char ctl_t::clim_oh_filename[LEN]`

Filename of OH climatology.

Definition at line 1113 of file libtrac.h.

4.4.2.139 clim_h2o2_filename `char ctl_t::clim_h2o2_filename[LEN]`

Filename of H2O2 climatology.

Definition at line 1116 of file libtrac.h.

4.4.2.140 oh_chem_reaction `int ctl_t::oh_chem_reaction`

Reaction type for OH chemistry (0=none, 2=bimolecular, 3=termolecular).

Definition at line 1119 of file libtrac.h.

4.4.2.141 oh_chem `double ctl_t::oh_chem[4]`

Coefficients for OH reaction rate (A, E/R or k0, n, kinf, m).

Definition at line 1122 of file libtrac.h.

4.4.2.142 oh_chem_beta `double ctl_t::oh_chem_beta`

Beta parameter for diurnal variablity of OH.

Definition at line 1125 of file libtrac.h.

4.4.2.143 h2o2_chem_cc `double ctl_t::h2o2_chem_cc`

Cloud cover parameter for H2O2 chemistry.

Definition at line 1128 of file libtrac.h.

4.4.2.144 h2o2_chem_reaction `int ctl_t::h2o2_chem_reaction`

Reaction type for H2O2 chemistry (0=none, 1=SO2).

Definition at line 1131 of file libtrac.h.

4.4.2.145 chemgrid_nz `int ctl_t::chemgrid_nz`

Number of altitudes of chemistry grid.

Definition at line 1134 of file libtrac.h.

4.4.2.146 chemgrid_z0 `double ctl_t::chemgrid_z0`

Lower altitude of chemistry grid [km].

Definition at line 1137 of file libtrac.h.

4.4.2.147 chemgrid_z1 `double ctl_t::chemgrid_z1`

Upper altitude of chemistry grid [km].

Definition at line 1140 of file libtrac.h.

4.4.2.148 chemgrid_nx `int ctl_t::chemgrid_nx`

Number of longitudes of chemistry grid.

Definition at line 1143 of file libtrac.h.

4.4.2.149 chemgrid_lon0 `double ctl_t::chemgrid_lon0`

Lower longitude of chemistry grid [deg].

Definition at line 1146 of file libtrac.h.

4.4.2.150 chemgrid_lon1 `double ctl_t::chemgrid_lon1`

Upper longitude of chemistry grid [deg].

Definition at line 1149 of file libtrac.h.

4.4.2.151 chemgrid_ny `int ctl_t::chemgrid_ny`

Number of latitudes of chemistry grid.

Definition at line 1152 of file libtrac.h.

4.4.2.152 chemgrid_lat0 `double ctl_t::chemgrid_lat0`

Lower latitude of chemistry grid [deg].

Definition at line 1155 of file libtrac.h.

4.4.2.153 chemgrid_lat1 `double ctl_t::chemgrid_lat1`

Upper latitude of chemistry grid [deg].

Definition at line 1158 of file libtrac.h.

4.4.2.154 dry_depo_dp `double ctl_t::dry_depo_dp`

Dry deposition surface layer [hPa].

Definition at line 1161 of file libtrac.h.

4.4.2.155 dry_depo_vdep `double ctl_t::dry_depo_vdep`

Dry deposition velocity [m/s].

Definition at line 1164 of file libtrac.h.

4.4.2.156 wet_depo_pre `double ctl_t::wet_depo_pre[2]`

Coefficients for precipitation calculation.

Definition at line 1167 of file libtrac.h.

4.4.2.157 wet_depo_bc_a `double ctl_t::wet_depo_bc_a`

Coefficient A for wet deposition below cloud (exponential form).

Definition at line 1170 of file libtrac.h.

4.4.2.158 wet_depo_bc_b `double ctl_t::wet_depo_bc_b`

Coefficient B for wet deposition below cloud (exponential form).

Definition at line 1173 of file libtrac.h.

4.4.2.159 wet_depo_ic_a `double ctl_t::wet_depo_ic_a`

Coefficient A for wet deposition in cloud (exponential form).

Definition at line 1176 of file libtrac.h.

4.4.2.160 wet_depo_ic_b `double ctl_t::wet_depo_ic_b`

Coefficient B for wet deposition in cloud (exponential form).

Definition at line 1179 of file libtrac.h.

4.4.2.161 wet_depo_ic_h `double ctl_t::wet_depo_ic_h[3]`

Coefficients for wet deposition in cloud (Henry's law: Hb, Cb, pH).

Definition at line 1182 of file libtrac.h.

4.4.2.162 wet_depo_bc_h `double ctl_t::wet_depo_bc_h[2]`

Coefficients for wet deposition below cloud (Henry's law: Hb, Cb).

Definition at line 1185 of file libtrac.h.

4.4.2.163 wet_depo_ic_ret_ratio `double ctl_t::wet_depo_ic_ret_ratio`

Coefficients for wet deposition in cloud: retention ratio.

Definition at line 1188 of file libtrac.h.

4.4.2.164 wet_depo_bc_ret_ratio `double ctl_t::wet_depo_bc_ret_ratio`

Coefficients for wet deposition below cloud: retention ratio.

Definition at line 1191 of file libtrac.h.

4.4.2.165 psc_h2o `double ctl_t::psc_h2o`

H2O volume mixing ratio for PSC analysis.

Definition at line 1194 of file libtrac.h.

4.4.2.166 psc_hno3 `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

Definition at line 1197 of file libtrac.h.

4.4.2.167 atm_basename `char ctl_t::atm_basename[LEN]`

Basename of atmospheric data files.

Definition at line 1200 of file libtrac.h.

4.4.2.168 atm_gpfile `char ctl_t::atm_gpfile[LEN]`

Gnuplot file for atmospheric data.

Definition at line 1203 of file libtrac.h.

4.4.2.169 atm_dt_out `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

Definition at line 1206 of file libtrac.h.

4.4.2.170 atm_filter `int ctl_t::atm_filter`

Time filter for atmospheric data output (0=none, 1=missval, 2=remove).

Definition at line 1209 of file libtrac.h.

4.4.2.171 atm_stride `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line 1212 of file libtrac.h.

4.4.2.172 atm_type `int ctl_t::atm_type`

Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF, 3=CLaMS).

Definition at line 1215 of file libtrac.h.

4.4.2.173 csi_basename `char ctl_t::csi_basename[LEN]`

Basename of CSI data files.

Definition at line 1218 of file libtrac.h.

4.4.2.174 csi_dt_out `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line 1221 of file libtrac.h.

4.4.2.175 csi_obsfile `char ctl_t::csi_obsfile[LEN]`

Observation data file for CSI analysis.

Definition at line 1224 of file libtrac.h.

4.4.2.176 csi_obsmin `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line 1227 of file libtrac.h.

4.4.2.177 csi_modmin `double ctl_t::csi_modmin`

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

Definition at line 1230 of file libtrac.h.

4.4.2.178 csi_nz `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line 1233 of file libtrac.h.

4.4.2.179 csi_z0 `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

Definition at line 1236 of file libtrac.h.

4.4.2.180 csi_z1 `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

Definition at line 1239 of file libtrac.h.

4.4.2.181 csi_nx `int ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

Definition at line 1242 of file libtrac.h.

4.4.2.182 csi_lon0 `double ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 1245 of file libtrac.h.

4.4.2.183 csi_lon1 `double ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 1248 of file libtrac.h.

4.4.2.184 csi_ny `int ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 1251 of file libtrac.h.

4.4.2.185 csi_lat0 `double ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 1254 of file libtrac.h.

4.4.2.186 csi_lat1 `double ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line 1257 of file libtrac.h.

4.4.2.187 ens_basename `char ctl_t::ens_basename[LEN]`

Basename of ensemble data file.

Definition at line 1260 of file libtrac.h.

4.4.2.188 ens_dt_out `double ctl_t::ens_dt_out`

Time step for ensemble output [s].

Definition at line 1263 of file libtrac.h.

4.4.2.189 grid_basename `char ctl_t::grid_basename[LEN]`

Basename of grid data files.

Definition at line 1266 of file libtrac.h.

4.4.2.190 grid_gpfile `char ctl_t::grid_gpfile[LEN]`

Gnuplot file for gridded data.

Definition at line 1269 of file libtrac.h.

4.4.2.191 grid_dt_out `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

Definition at line 1272 of file libtrac.h.

4.4.2.192 grid_sparse `int ctl_t::grid_sparse`

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

Definition at line 1275 of file libtrac.h.

4.4.2.193 grid_nz `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line 1278 of file libtrac.h.

4.4.2.194 grid_z0 `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line 1281 of file libtrac.h.

4.4.2.195 grid_z1 `double ctl_t::grid_z1`

Upper altitude of gridded data [km].

Definition at line 1284 of file libtrac.h.

4.4.2.196 grid_nx `int ctl_t::grid_nx`

Number of longitudes of gridded data.

Definition at line 1287 of file libtrac.h.

4.4.2.197 grid_lon0 `double ctl_t::grid_lon0`

Lower longitude of gridded data [deg].

Definition at line 1290 of file libtrac.h.

4.4.2.198 grid_lon1 `double ctl_t::grid_lon1`

Upper longitude of gridded data [deg].

Definition at line 1293 of file libtrac.h.

4.4.2.199 grid_ny `int ctl_t::grid_ny`

Number of latitudes of gridded data.

Definition at line 1296 of file libtrac.h.

4.4.2.200 grid_lat0 `double ctl_t::grid_lat0`

Lower latitude of gridded data [deg].

Definition at line 1299 of file libtrac.h.

4.4.2.201 grid_lat1 `double ctl_t::grid_lat1`

Upper latitude of gridded data [deg].

Definition at line 1302 of file libtrac.h.

4.4.2.202 grid_type `int ctl_t::grid_type`

Type of grid data files (0=ASCII, 1=netCDF).

Definition at line 1305 of file libtrac.h.

4.4.2.203 prof_basename `char ctl_t::prof_basename[LEN]`

Basename for profile output file.

Definition at line 1308 of file libtrac.h.

4.4.2.204 prof_obsfile `char ctl_t::prof_obsfile[LEN]`

Observation data file for profile output.

Definition at line 1311 of file libtrac.h.

4.4.2.205 prof_nz `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line 1314 of file libtrac.h.

4.4.2.206 prof_z0 `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line 1317 of file libtrac.h.

4.4.2.207 prof_z1 `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line 1320 of file libtrac.h.

4.4.2.208 prof_nx `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line 1323 of file libtrac.h.

4.4.2.209 prof_lon0 `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line 1326 of file libtrac.h.

4.4.2.210 prof_lon1 `double ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line 1329 of file libtrac.h.

4.4.2.211 prof_ny `int ctl_t::prof_ny`

Number of latitudes of gridded profile data.

Definition at line 1332 of file libtrac.h.

4.4.2.212 prof_lat0 `double ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

Definition at line 1335 of file libtrac.h.

4.4.2.213 prof_lat1 `double ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

Definition at line 1338 of file libtrac.h.

4.4.2.214 sample_basename `char ctl_t::sample_basename[LEN]`

Basename of sample data file.

Definition at line 1341 of file libtrac.h.

4.4.2.215 sample_obsfile `char ctl_t::sample_obsfile[LEN]`

Observation data file for sample output.

Definition at line 1344 of file libtrac.h.

4.4.2.216 sample_dx `double ctl_t::sample_dx`

Horizontal radius for sample output [km].

Definition at line 1347 of file libtrac.h.

4.4.2.217 sample_dz `double ctl_t::sample_dz`

Layer depth for sample output [km].

Definition at line 1350 of file libtrac.h.

4.4.2.218 stat_basename `char ctl_t::stat_basename[LEN]`

Basename of station data file.

Definition at line 1353 of file libtrac.h.

4.4.2.219 stat_lon `double ctl_t::stat_lon`

Longitude of station [deg].

Definition at line 1356 of file libtrac.h.

4.4.2.220 stat_lat `double ctl_t::stat_lat`

Latitude of station [deg].

Definition at line 1359 of file libtrac.h.

4.4.2.221 stat_r `double ctl_t::stat_r`

Search radius around station [km].

Definition at line 1362 of file libtrac.h.

4.4.2.222 stat_t0 `double ctl_t::stat_t0`

Start time for station output [s].

Definition at line 1365 of file libtrac.h.

4.4.2.223 stat_t1 `double ctl_t::stat_t1`

Stop time for station output [s].

Definition at line 1368 of file libtrac.h.

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

- libtrac.h

4.5 met_t Struct Reference

Meteo 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 ts [EX][EY]
  Surface temperature [K].
- float zs [EX][EY]
  Surface geopotential height [km].
- float us [EX][EY]
  Surface zonal wind [m/s].
- float vs [EX][EY]
  Surface meridional wind [m/s].
- float pbl [EX][EY]
  Boundary layer pressure [hPa].
- float pt [EX][EY]
  Tropopause pressure [hPa].
- float tt [EX][EY]
  Tropopause temperature [K].
- float zt [EX][EY]
  Tropopause geopotential height [km].
- float h2ot [EX][EY]
  Tropopause water vapor vmr [ppv].
- float pct [EX][EY]
  Cloud top pressure [hPa].
- float pcb [EX][EY]
  Cloud bottom pressure [hPa].
- float cl [EX][EY]
  Total column cloud water [kg/m^2].
- float plcl [EX][EY]
  Pressure at lifted condensation level (LCL) [hPa].
- float plfc [EX][EY]
  Pressure at level of free convection (LFC) [hPa].
- float pel [EX][EY]
  Pressure at equilibrium level [hPa].
- float cape [EX][EY]
  Convective available potential energy [J/kg].
- float cin [EX][EY]

Convective inhibition [J/kg].

- float z [EX][EY][EP]

  Geopotential height [km].
- float t [EX][EY][EP]

  Temperature [K].
- float u [EX][EY][EP]

  Zonal wind [m/s].
- float v [EX][EY][EP]

  Meridional wind [m/s].
- float w [EX][EY][EP]

  Vertical velocity [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].
- float patp [EX][EY][EP]

  Pressure field in pressure levels [hPa].
- float zeta [EX][EY][EP]

  Zeta [K].
- float zeta_dot [EX][EY][EP]

  Vertical velocity [K/s].

4.5.1 Detailed Description

Meteo data.

Definition at line 1502 of file libtrac.h.

4.5.2 Field Documentation

4.5.2.1 time `double met_t::time`

Time [s].

Definition at line 1505 of file libtrac.h.

4.5.2.2 nx `int met_t::nx`

Number of longitudes.

Definition at line 1508 of file libtrac.h.

4.5.2.3 ny `int met_t::ny`

Number of latitudes.

Definition at line 1511 of file libtrac.h.

4.5.2.4 np `int met_t::np`

Number of pressure levels.

Definition at line 1514 of file libtrac.h.

4.5.2.5 lon `double met_t::lon[EX]`

Longitude [deg].

Definition at line 1517 of file libtrac.h.

4.5.2.6 lat `double met_t::lat[EY]`

Latitude [deg].

Definition at line 1520 of file libtrac.h.

4.5.2.7 p `double met_t::p[EP]`

Pressure [hPa].

Definition at line 1523 of file libtrac.h.

4.5.2.8 ps `float met_t::ps[EX][EY]`

Surface pressure [hPa].

Definition at line 1526 of file libtrac.h.

4.5.2.9 ts `float met_t::ts[EX][EY]`

Surface temperature [K].

Definition at line 1529 of file libtrac.h.

4.5.2.10 zs `float met_t::zs[EX][EY]`

Surface geopotential height [km].

Definition at line 1532 of file libtrac.h.

4.5.2.11 us `float met_t::us[EX][EY]`

Surface zonal wind [m/s].

Definition at line 1535 of file libtrac.h.

4.5.2.12 vs `float met_t::vs[EX][EY]`

Surface meridional wind [m/s].

Definition at line 1538 of file libtrac.h.

4.5.2.13 pbl `float met_t::pbl[EX][EY]`

Boundary layer pressure [hPa].

Definition at line 1541 of file libtrac.h.

4.5.2.14 pt `float met_t::pt[EX][EY]`

Tropopause pressure [hPa].

Definition at line 1544 of file libtrac.h.

4.5.2.15 tt `float met_t::tt[EX][EY]`

Tropopause temperature [K].

Definition at line 1547 of file libtrac.h.

4.5.2.16 zt `float met_t::zt[EX][EY]`

Tropopause geopotential height [km].

Definition at line 1550 of file libtrac.h.

4.5.2.17 h2ot `float met_t::h2ot[EX][EY]`

Tropopause water vapor vmr [ppv].

Definition at line 1553 of file libtrac.h.

4.5.2.18 pct `float met_t::pct[EX][EY]`

Cloud top pressure [hPa].

Definition at line 1556 of file libtrac.h.

4.5.2.19 pcb `float met_t::pcb[EX][EY]`

Cloud bottom pressure [hPa].

Definition at line 1559 of file libtrac.h.

4.5.2.20 cl `float met_t::cl[EX][EY]`

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

Definition at line 1562 of file libtrac.h.

4.5.2.21 plcl `float met_t::plcl[EX][EY]`

Pressure at lifted condensation level (LCL) [hPa].

Definition at line 1565 of file libtrac.h.

4.5.2.22 plfc `float met_t::plfc[EX][EY]`

Pressure at level of free convection (LFC) [hPa].

Definition at line 1568 of file libtrac.h.

4.5.2.23 pel `float met_t::pel[EX][EY]`

Pressure at equilibrium level [hPa].

Definition at line 1571 of file libtrac.h.

4.5.2.24 cape `float met_t::cape[EX][EY]`

Convective available potential energy [J/kg].

Definition at line 1574 of file libtrac.h.

4.5.2.25 cin `float met_t::cin[EX][EY]`

Convective inhibition [J/kg].

Definition at line 1577 of file libtrac.h.

4.5.2.26 z `float met_t::z[EX][EY][EP]`

Geopotential height [km].

Definition at line 1580 of file libtrac.h.

4.5.2.27 t `float met_t::t[EX][EY][EP]`

Temperature [K].

Definition at line 1583 of file libtrac.h.

4.5.2.28 u `float met_t::u[EX][EY][EP]`

Zonal wind [m/s].

Definition at line 1586 of file libtrac.h.

4.5.2.29 v `float met_t::v[EX][EY][EP]`

Meridional wind [m/s].

Definition at line 1589 of file libtrac.h.

4.5.2.30 w `float met_t::w[EX][EY][EP]`

Vertical velocity [hPa/s].

Definition at line 1592 of file libtrac.h.

4.5.2.31 pv `float met_t::pv[EX][EY][EP]`

Potential vorticity [PVU].

Definition at line 1595 of file libtrac.h.

4.5.2.32 h2o `float met_t::h2o[EX][EY][EP]`

Water vapor volume mixing ratio [1].

Definition at line 1598 of file libtrac.h.

4.5.2.33 o3 `float met_t::o3[EX][EY][EP]`

Ozone volume mixing ratio [1].

Definition at line 1601 of file libtrac.h.

4.5.2.34 lwc `float met_t::lwc[EX][EY][EP]`

Cloud liquid water content [kg/kg].

Definition at line 1604 of file libtrac.h.

4.5.2.35 iwc `float met_t::iwc[EX][EY][EP]`

Cloud ice water content [kg/kg].

Definition at line 1607 of file libtrac.h.

4.5.2.36 pl `float met_t::pl[EX][EY][EP]`

Pressure on model levels [hPa].

Definition at line 1610 of file libtrac.h.

4.5.2.37 patp `float met_t::patp[EX][EY][EP]`

Pressure field in pressure levels [hPa].

Definition at line 1613 of file libtrac.h.

4.5.2.38 zeta `float met_t::zeta[EX][EY][EP]`

Zeta [K].

Definition at line 1616 of file libtrac.h.

4.5.2.39 zeta_dot `float met_t::zeta_dot[EX][EY][EP]`

Vertical velocity [K/s].

Definition at line 1619 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.

```
#include "libtrac.h"
```

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

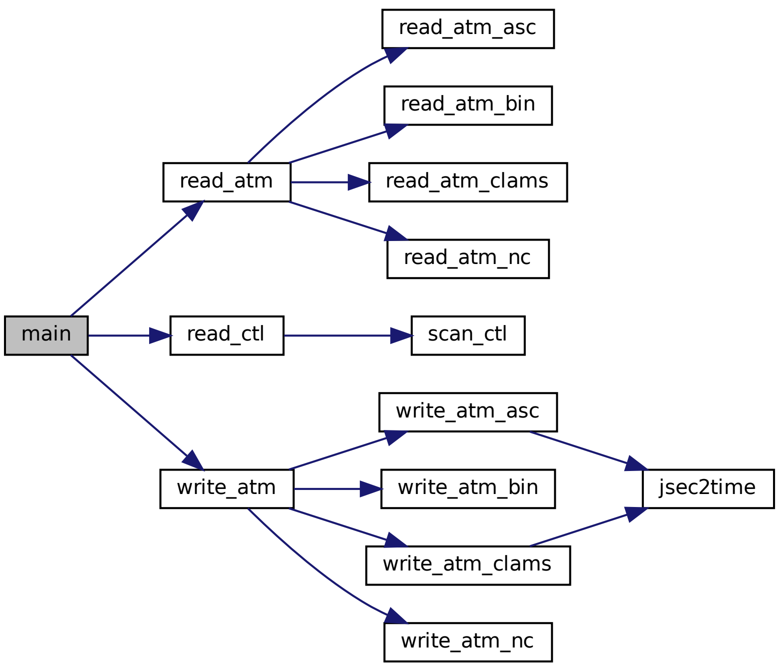
5.1.2.1 main()

```
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) 2013-2022 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.

```
#include "libtrac.h"
```

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 main()

```
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, zscore;

  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);
  zscore = scan_ctl(argv[1], argc, argv, "DIST_ZSCORE", -1, "-999", NULL);

  /* Write info... */
```

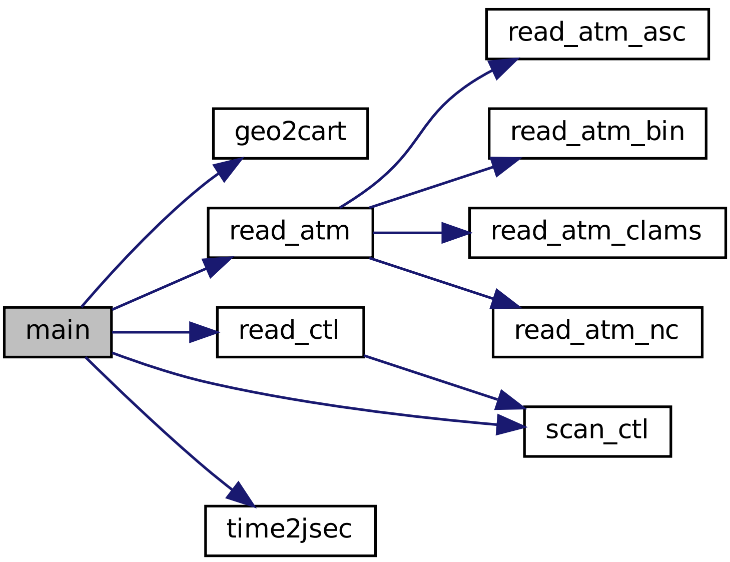
```
  LOG(1, "Write transport deviations: %s", argv[2]);

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

    /* Check time... */
    if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
        || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
      ERRMSG("Cannot read time from filename!");

    /* 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 air parcel index... */
      if (ctl.qnt_idx > 0
          && (atm1->q[ctl.qnt_idx][ip] != atm2->q[ctl.qnt_idx][ip]))
        ERRMSG("Air parcel index does not match!");

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

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

    /* Filter data... */
    if (zscore > 0 && np > 1) {

      /* Get means and standard deviations of transport deviations... */
      size_t n = (size_t) np;
      double muh = gsl_stats_mean(ahtd, 1, n);
      double muv = gsl_stats_mean(avtd, 1, n);
      double sigh = gsl_stats_sd(ahtd, 1, n);
      double sigv = gsl_stats_sd(avtd, 1, n);

      /* Filter data... */
      np = 0;
      for (size_t i = 0; i < n; i++)
        if (fabs((ahtd[i] - muh) / sigh) < zscore
            && fabs((avtd[i] - muv) / sigv) < zscore) {
          ahtd[np] = ahtd[i];
          rhtd[np] = rhtd[i];
          avtd[np] = avtd[i];
          rvtd[np] = rvtd[i];
          for (iq = 0; iq < ctl.nq; iq++) {
            aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
            rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
          }
          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], "absdev") == 0) {
    ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
    rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
    avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
    rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
    for (iq = 0; iq < ctl.nq; iq++) {
      aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
      rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
    }
  } 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], "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) 2013-2022 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, zscore;

  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);
  zscore = scan_ctl(argv[1], argc, argv, "DIST_ZSCORE", -1, "-999", NULL);
```

```
/* Write info... */
LOG(1, "Write transport deviations: %s", 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... */
  size_t len = strlen(argv[f]);
  sprintf(tstr, "%.4s", &argv[f][len - 20]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 15]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 12]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 9]);
  hour = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 6]);
  min = atoi(tstr);
  time2jsec(year, mon, day, hour, min, 0, 0, &t);

  /* Check time... */
  if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
      || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
    ERRMSG("Cannot read time from filename!");

  /* 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 air parcel index... */
    if (ctl.qnt_idx > 0
        && (atm1->q[ctl.qnt_idx][ip] != atm2->q[ctl.qnt_idx][ip]))
      ERRMSG("Air parcel index does not match!");

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

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

    /* Filter data... */
    if (zscore > 0 && np > 1) {

      /* Get means and standard deviations of transport deviations... */
      size_t n = (size_t) np;
      double muh = gsl_stats_mean(ahtd, 1, n);
      double muv = gsl_stats_mean(avtd, 1, n);
      double sigh = gsl_stats_sd(ahtd, 1, n);
      double sigv = gsl_stats_sd(avtd, 1, n);

      /* Filter data... */
      np = 0;
      for (size_t i = 0; i < n; i++)
        if (fabs((ahtd[i] - muh) / sigh) < zscore
            && fabs((avtd[i] - muv) / sigv) < zscore) {
          ahtd[np] = ahtd[i];
          rhtd[np] = rhtd[i];
          avtd[np] = avtd[i];
          rvtd[np] = rvtd[i];
          for (iq = 0; iq < ctl.nq; iq++) {
            aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
            rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
          }
          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], "absdev") == 0) {
      ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
      rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
      avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
      rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
      for (iq = 0; iq < ctl.nq; iq++) {
        aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
        rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
      }
    } 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], "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.

```
#include "libtrac.h"
```

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 main()

```
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, vmr, bellrad;

  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);
  vmr = scan_ctl(argv[1], argc, argv, "INIT_VMR", -1, "0", NULL);
  bellrad = scan_ctl(argv[1], argc, argv, "INIT_BELLRAD", -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.)));

            /* Apply cosine bell (Williamson et al., 1992)... */
```

```
            if (bellrad > 0) {
              double x0[3], x1[3];
              geo2cart(0.0, 0.5 * (lon0 + lon1), 0.5 * (lat0 + lat1), x0);
              geo2cart(0.0, atm->lon[atm->np], atm->lat[atm->np], x1);
              double rad = RE * acos(DOTP(x0, x1) / NORM(x0) / NORM(x1));
              if (rad > bellrad)
                continue;
              if (ctl.qnt_m >= 0)
                atm->q[ctl.qnt_m][atm->np] =
                  0.5 * (1. + cos(M_PI * rad / bellrad));
              if (ctl.qnt_vmr >= 0)
                atm->q[ctl.qnt_vmr][atm->np] =
                  0.5 * (1. + cos(M_PI * rad / bellrad));
            }

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

  /* Initialize volume mixing ratio... */
  if (ctl.qnt_vmr >= 0 && bellrad <= 0)
    for (ip = 0; ip < atm->np; ip++)
      atm->q[ctl.qnt_vmr][ip] = vmr;

  /* Initialize air parcel index... */
  if (ctl.qnt_idx >= 0)
    for (ip = 0; ip < atm->np; ip++)
      atm->q[ctl.qnt_idx][ip] = ip;

  /* Save data... */
  write_atm(argv[2], &ctl, atm, 0);

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

  return EXIT_SUCCESS;
}
```

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) 2013-2022 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, vmr, bellrad;
```

```
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);
vmr = scan_ctl(argv[1], argc, argv, "INIT_VMR", -1, "0", NULL);
bellrad = scan_ctl(argv[1], argc, argv, "INIT_BELLRAD", -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.)));

          /* Apply cosine bell (Williamson et al., 1992)... */
          if (bellrad > 0) {
            double x0[3], x1[3];
            geo2cart(0.0, 0.5 * (lon0 + lon1), 0.5 * (lat0 + lat1), x0);
            geo2cart(0.0, atm->lon[atm->np], atm->lat[atm->np], x1);
            double rad = RE * acos(DOTP(x0, x1) / NORM(x0) / NORM(x1));
            if (rad > bellrad)
              continue;
            if (ctl.qnt_m >= 0)
              atm->q[ctl.qnt_m][atm->np] =
                0.5 * (1. + cos(M_PI * rad / bellrad));
            if (ctl.qnt_vmr >= 0)
              atm->q[ctl.qnt_vmr][atm->np] =
                0.5 * (1. + cos(M_PI * rad / bellrad));
          }

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

    /* Initialize volume mixing ratio... */
    if (ctl.qnt_vmr >= 0 && bellrad <= 0)
      for (ip = 0; ip < atm->np; ip++)
        atm->q[ctl.qnt_vmr][ip] = vmr;

    /* Initialize air parcel index... */
    if (ctl.qnt_idx >= 0)
      for (ip = 0; ip < atm->np; ip++)
        atm->q[ctl.qnt_idx][ip] = ip;

    /* 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.

```
#include "libtrac.h"
```

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

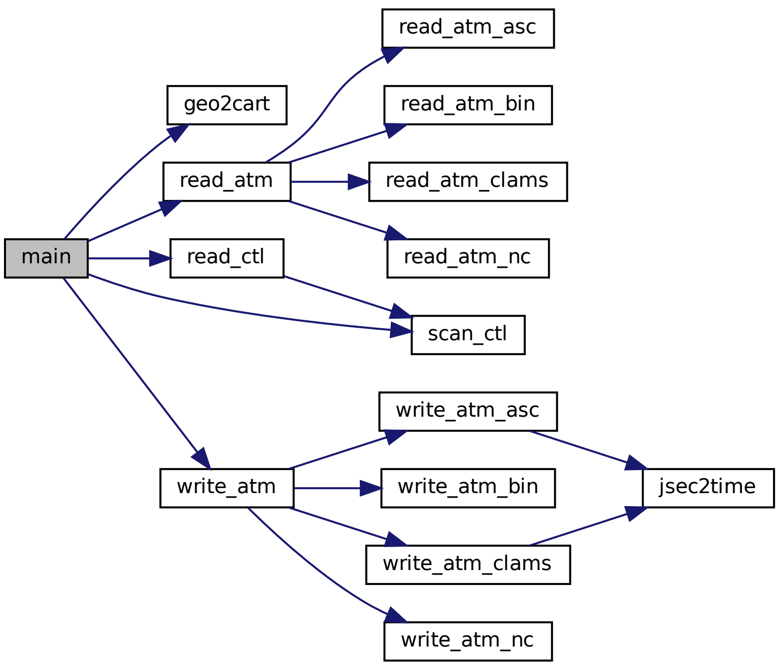
5.7.2.1 main() `int main ( int argc, char * argv[] )`

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, idx0, idx1, 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);
  idx0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX0", -1, "-999", NULL);
  idx1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX1", -1, "-999", NULL);
  ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
  ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", 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;

    /* Adjust range of air parcels... */
    if (ip0 < 0)
      ip0 = 0;
    ip0 = GSL_MIN(ip0, atm->np - 1);
    if (ip1 < 0)
      ip1 = atm->np - 1;
    ip1 = GSL_MIN(ip1, atm->np - 1);
    if (ip1 < ip0)
      ip1 = ip0;

    /* Loop over air parcels... */
    for (ip = ip0; ip <= ip1; ip += stride) {

      /* Check air parcel index... */
      if (ctl.qnt_idx >= 0 && idx0 >= 0 && idx1 >= 0)
        if (atm->q[ctl.qnt_idx][ip] < idx0 || atm->q[ctl.qnt_idx][ip] > idx1)
          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) 2013-2022 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, idx0, idx1, 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);
  idx0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX0", -1, "-999", NULL);
  idx1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX1", -1, "-999", NULL);
  ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
  ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", 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;

    /* Adjust range of air parcels... */
    if (ip0 < 0)
      ip0 = 0;
    ip0 = GSL_MIN(ip0, atm->np - 1);
    if (ip1 < 0)
      ip1 = atm->np - 1;
    ip1 = GSL_MIN(ip1, atm->np - 1);
    if (ip1 < ip0)
      ip1 = ip0;

    /* Loop over air parcels... */
    for (ip = ip0; ip <= ip1; ip += stride) {
```

```
      /* Check air parcel index... */
      if (ctl.qnt_idx >= 0 && idx0 >= 0 && idx1 >= 0)
        if (atm->q[ctl.qnt_idx][ip] < idx0 || atm->q[ctl.qnt_idx][ip] > idx1)
          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.

```
#include "libtrac.h"
```

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 main()

```
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[EP], kz[EP], 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... */
    LOG(1, "Read kernel function: %s", 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) >= EP)
          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... */
    do {
      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]);
    } while (atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));

    /* 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;

    /* Adjust air parcel index... */
    if (ctl.qnt_idx >= 0)
      atm2->q[ctl.qnt_idx][atm2->np] = atm2->np;

    /* 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;
}
```

5.10 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) 2013-2022 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[EP], kz[EP], 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... */
    LOG(1, "Read kernel function: %s", 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) >= EP)
          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... */
    do {
      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]);
    } while (atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));

    /* 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;

    /* Adjust air parcel index... */
    if (ctl.qnt_idx >= 0)
      atm2->q[ctl.qnt_idx][atm2->np] = atm2->np;

    /* 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.

```
#include "libtrac.h"
```

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 main()

```
int main (
    int argc,
    char ∗ argv[ ] )
```

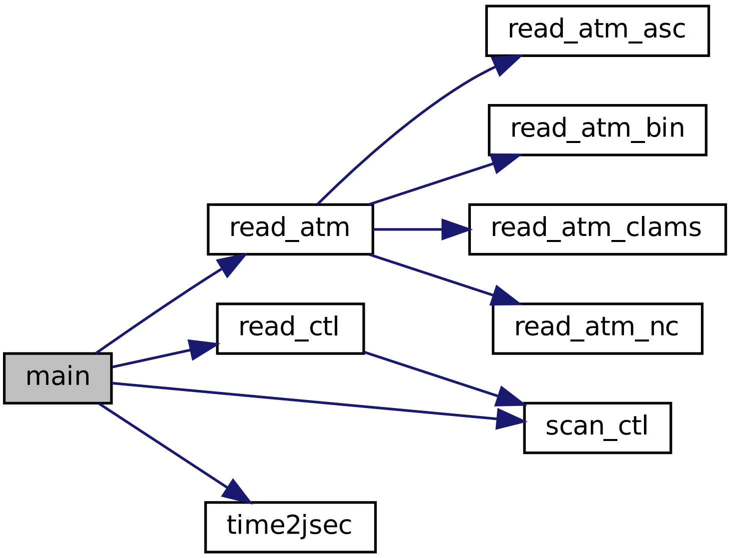
Definition at line 27 of file atm_stat.c.

```
00029                    {
00030 
00031   ctl_t ctl;
00032 
00033   atm_t *atm, *atm_filt;
00034 
00035   FILE *out;
00036 
00037   char tstr[LEN];
00038 
00039   double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040     t, t0 = GSL_NAN, qm[NQ], *work, zm, *zs;
00041 
00042   int ens, f, init = 0, ip, iq, year, mon, day, hour, min;
00043 
00044   /* Allocate... */
00045   ALLOC(atm, atm_t, 1);
00046   ALLOC(atm_filt, atm_t, 1);
00047   ALLOC(work, double,
00048         NP);
00049   ALLOC(zs, double,
00050         NP);
00051 
00052   /* Check arguments... */
00053   if (argc < 4)
00054     ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
```

```
  /* 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... */
  LOG(1, "Write air parcel statistics: %s", 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... */
    size_t len = strlen(argv[f]);
    sprintf(tstr, "%.4s", &argv[f][len - 20]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][len - 15]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][len - 12]);
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][len - 9]);
    hour = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][len - 6]);
    min = atoi(tstr);
    time2jsec(year, mon, day, hour, min, 0, 0, &t);

    /* Check time... */
    if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
        || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
      ERRMSG("Cannot read time from filename!");

    /* Save initial 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) 2013-2022 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 = GSL_NAN, 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... */
LOG(1, "Write air parcel statistics: %s", 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... */
  size_t len = strlen(argv[f]);
  sprintf(tstr, "%.4s", &argv[f][len - 20]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 15]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 12]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 9]);
  hour = atoi(tstr);
  sprintf(tstr, "%.2s", &argv[f][len - 6]);
  min = atoi(tstr);
  time2jsec(year, mon, day, hour, min, 0, 0, &t);

  /* Check time... */
  if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
      || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
    ERRMSG("Cannot read time from filename!");

  /* Save initial 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 cape.c File Reference

Add CAPE data to netCDF file.

`#include "libtrac.h"`

Functions

- int main (int argc, char ∗argv[ ])

5.13.1 Detailed Description

Add CAPE data to netCDF file.

Definition in file cape.c.

5.13.2 Function Documentation

5.13.2.1 main()

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

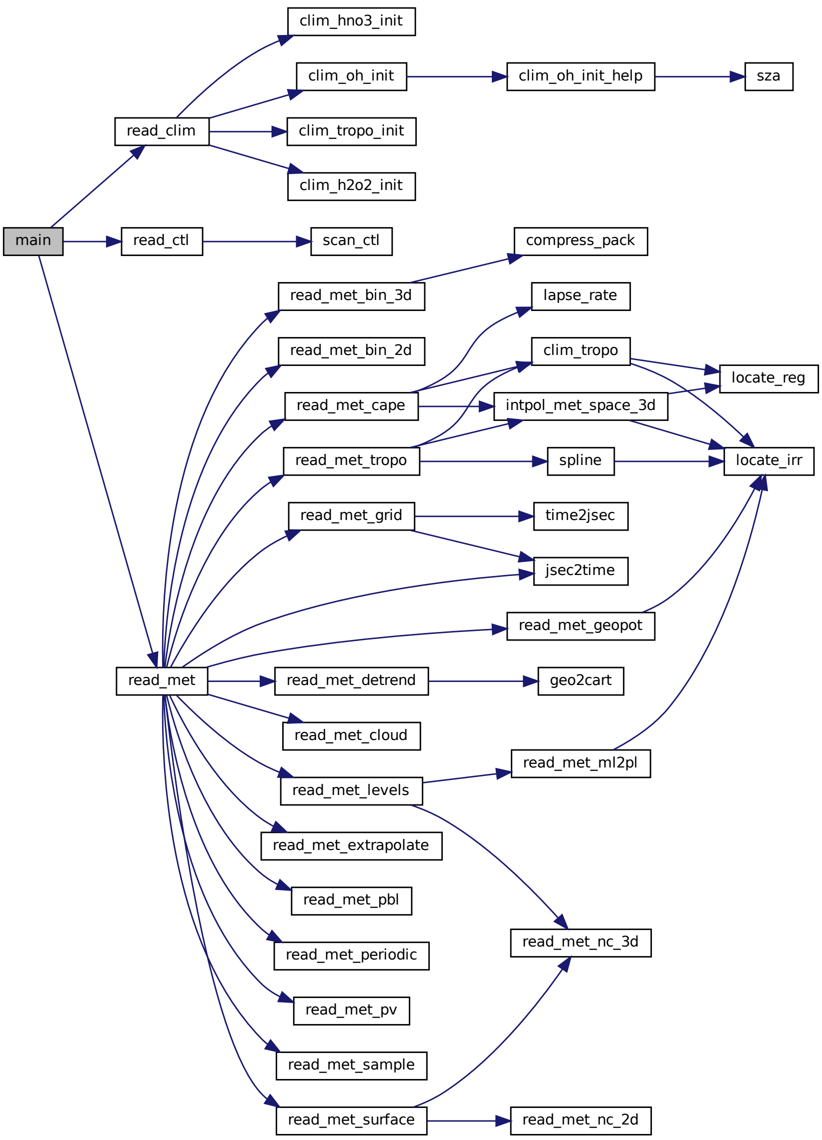
Definition at line 27 of file cape.c.

```
                    {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  char tstr[LEN];

  float help[EX * EY];

  int dims[10], ncid, varid;

  size_t start[10], count[10];

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

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

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

  /* Read climatological data... */
```

```
  read_clim(&ctl, clim);

  /* Read meteorological data... */
  if (!read_met(argv[2], &ctl, clim, met))
    ERRMSG("Cannot open file!");

  /* Open netCDF file... */
  if (nc_open(argv[2], NC_WRITE, &ncid) != NC_NOERR)
    ERRMSG("Cannot open file!");

  /* Get dimensions... */
  NC_INQ_DIM("time", &dims[0], 1, 1);
  NC_INQ_DIM("lat", &dims[1], met->ny, met->ny);
  NC_INQ_DIM("lon", &dims[2], met->nx - 1, met->nx - 1);
  NC(nc_inq_dimid(ncid, "time", &dims[0]));
  NC(nc_inq_dimid(ncid, "lat", &dims[1]));
  NC(nc_inq_dimid(ncid, "lon", &dims[2]));

  /* Set define mode... */
  NC(nc_redef(ncid));

  /* Create variables... */
  NC_DEF_VAR("CAPE_MPT", NC_FLOAT, 3, dims,
             "convective available potential energy", "J kg**-1");
  NC_DEF_VAR("CIN_MPT", NC_FLOAT, 3, dims,
             "convective inhibition", "J kg**-1");
  NC_DEF_VAR("PEL_MPT", NC_FLOAT, 3, dims,
             "pressure at equilibrium level", "hPa");

  /* Get current time... */
  time_t t = time(NULL);
  struct tm tm = *localtime(&t);
  sprintf(tstr, "%d-%02d-%02d %02d:%02d:%02d",
          tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
          tm.tm_hour, tm.tm_min, tm.tm_sec);

  /* Set additional attributes... */
  NC_PUT_ATT("CAPE_MPT", "creator_of_parameter", "MPTRAC");
  NC_PUT_ATT("CIN_MPT", "creator_of_parameter", "MPTRAC");
  NC_PUT_ATT("PEL_MPT", "creator_of_parameter", "MPTRAC");

  NC_PUT_ATT("CAPE_MPT", "param_creation_time", tstr);
  NC_PUT_ATT("CIN_MPT", "param_creation_time", tstr);
  NC_PUT_ATT("PEL_MPT", "param_creation_time", tstr);

  NC_PUT_ATT("CAPE_MPT", "param_modification_time", tstr);
  NC_PUT_ATT("CIN_MPT", "param_modification_time", tstr);
  NC_PUT_ATT("PEL_MPT", "param_modification_time", tstr);

  NC_PUT_ATT("CAPE_MPT", "flag", "NONE");
  NC_PUT_ATT("CIN_MPT", "flag", "NONE");
  NC_PUT_ATT("PEL_MPT", "flag", "NONE");

  float miss[1] = { GSL_NAN };
  NC(nc_inq_varid(ncid, "CAPE_MPT", &varid));
  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
  NC(nc_inq_varid(ncid, "CIN_MPT", &varid));
  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
  NC(nc_inq_varid(ncid, "PEL_MPT", &varid));
  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));

  /* End define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  for (int ix = 0; ix < met->nx - 1; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cape[ix][iy];
  NC_PUT_FLOAT("CAPE_MPT", help, 0);

  for (int ix = 0; ix < met->nx - 1; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cin[ix][iy];
  NC_PUT_FLOAT("CIN_MPT", help, 0);

  for (int ix = 0; ix < met->nx - 1; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(iy, ix, met->nx - 1)] = met->pel[ix][iy];
  NC_PUT_FLOAT("PEL_MPT", help, 0);

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

  /* Free... */
  free(clim);
  free(met);

```

```
  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.14 cape.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) 2013-2022 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  char tstr[LEN];

  float help[EX * EY];

  int dims[10], ncid, varid;

  size_t start[10], count[10];

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

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

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

  /* Read climatological data... */
  read_clim(&ctl, clim);

  /* Read meteorological data... */
  if (!read_met(argv[2], &ctl, clim, met))
    ERRMSG("Cannot open file!");

  /* Open netCDF file... */
  if (nc_open(argv[2], NC_WRITE, &ncid) != NC_NOERR)
    ERRMSG("Cannot open file!");

  /* Get dimensions... */
  NC_INQ_DIM("time", &dims[0], 1, 1);
  NC_INQ_DIM("lat", &dims[1], met->ny, met->ny);
  NC_INQ_DIM("lon", &dims[2], met->nx - 1, met->nx - 1);
  NC(nc_inq_dimid(ncid, "time", &dims[0]));
  NC(nc_inq_dimid(ncid, "lat", &dims[1]));
  NC(nc_inq_dimid(ncid, "lon", &dims[2]));

  /* Set define mode... */
  NC(nc_redef(ncid));

  /* Create variables... */
  NC_DEF_VAR("CAPE_MPT", NC_FLOAT, 3, dims,
             "convective available potential energy", "J kg**-1");
  NC_DEF_VAR("CIN_MPT", NC_FLOAT, 3, dims,
             "convective inhibition", "J kg**-1");
  NC_DEF_VAR("PEL_MPT", NC_FLOAT, 3, dims,
             "pressure at equilibrium level", "hPa");

  /* Get current time... */
  time_t t = time(NULL);
  struct tm tm = *localtime(&t);
  sprintf(tstr, "%d-%02d-%02d %02d:%02d:%02d",
          tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
          tm.tm_hour, tm.tm_min, tm.tm_sec);

  /* Set additional attributes... */
  NC_PUT_ATT("CAPE_MPT", "creator_of_parameter", "MPTRAC");
  NC_PUT_ATT("CIN_MPT", "creator_of_parameter", "MPTRAC");
  NC_PUT_ATT("PEL_MPT", "creator_of_parameter", "MPTRAC");

  NC_PUT_ATT("CAPE_MPT", "param_creation_time", tstr);
  NC_PUT_ATT("CIN_MPT", "param_creation_time", tstr);
  NC_PUT_ATT("PEL_MPT", "param_creation_time", tstr);

  NC_PUT_ATT("CAPE_MPT", "param_modification_time", tstr);
  NC_PUT_ATT("CIN_MPT", "param_modification_time", tstr);
  NC_PUT_ATT("PEL_MPT", "param_modification_time", tstr);

  NC_PUT_ATT("CAPE_MPT", "flag", "NONE");
  NC_PUT_ATT("CIN_MPT", "flag", "NONE");
```

```
  NC_PUT_ATT("PEL_MPT", "flag", "NONE");

  float miss[1] = { GSL_NAN };
  NC(nc_inq_varid(ncid, "CAPE_MPT", &varid));
  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
  NC(nc_inq_varid(ncid, "CIN_MPT", &varid));
  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
  NC(nc_inq_varid(ncid, "PEL_MPT", &varid));
  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));

  /* End define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  for (int ix = 0; ix < met->nx - 1; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cape[ix][iy];
  NC_PUT_FLOAT("CAPE_MPT", help, 0);

  for (int ix = 0; ix < met->nx - 1; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cin[ix][iy];
  NC_PUT_FLOAT("CIN_MPT", help, 0);

  for (int ix = 0; ix < met->nx - 1; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(iy, ix, met->nx - 1)] = met->pel[ix][iy];
  NC_PUT_FLOAT("PEL_MPT", help, 0);

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

5.15 day2doy.c File Reference

Convert date to day of year.

`#include "libtrac.h"`

Functions

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

5.15.1 Detailed Description

Convert date to day of year.

Definition in file day2doy.c.

5.15.2 Function Documentation

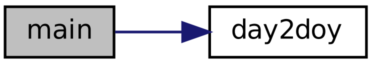
5.15.2.1 main()

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

Definition at line 27 of file day2doy.c.

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

Here is the call graph for this function:



5.16 day2doy.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   int day, doy, mon, year;
00032
00033   /* Check arguments... */
00034   if (argc < 4)
00035     ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037   /* Read arguments... */
00038   year = atoi(argv[1]);
00039   mon = atoi(argv[2]);
00040   day = atoi(argv[3]);
00041
00042   /* Convert... */
```

```
  day2doy(year, mon, day, &doy);
  printf("%d %d\n", year, doy);

  return EXIT_SUCCESS;
}
```

5.17 doy2day.c File Reference

Convert day of year to date.

`#include "libtrac.h"`

Functions

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

5.17.1 Detailed Description

Convert day of year to date.

Definition in file doy2day.c.

5.17.2 Function Documentation

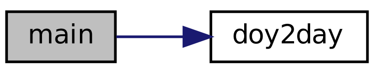
5.17.2.1 main()

```
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:



5.18 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;
}
```

5.19 jsec2time.c File Reference

Convert Julian seconds to date.

```
#include "libtrac.h"
```

Functions

- int main (int argc, char ∗argv[ ])

5.19.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

5.19.2 Function Documentation

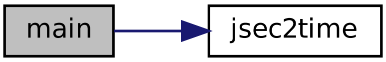
5.19.2.1 main()

```
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:



5.20 jsec2time.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 2)
00037     ERRMSG("Give parameters: <jsec>");
00038
00039   /* Read arguments... */
00040   jsec = atof(argv[1]);
00041
00042   /* Convert time... */
```

```
  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;
}
```

5.21 libtrac.c File Reference

MPTRAC library definitions.

```
#include "libtrac.h"
```

Functions

- double buoyancy_frequency (double p0, double t0, double p1, double t1)
  Calculate buoyancy frequency.
- void cart2geo (double ∗x, double ∗z, double ∗lon, double ∗lat)
  Convert Cartesian coordinates to geolocation.
- double clim_hno3 (clim_t ∗clim, double t, double lat, double p)
  Climatology of HNO3 volume mixing ratios.
- void clim_hno3_init (clim_t ∗clim)
  Initialization function for HNO3 climatology.
- double clim_oh (clim_t ∗clim, double t, double lat, double p)
  Climatology of OH number concentrations.
- double clim_oh_diurnal (ctl_t ∗ctl, clim_t ∗clim, double t, double p, double lon, double lat)
  Climatology of OH number concentrations with diurnal variation.
- void clim_oh_init (ctl_t ∗ctl, clim_t ∗clim)
  Initialization function for OH climatology.
- double clim_oh_init_help (double beta, double time, double lat)
  Apply diurnal correction to OH climatology.
- double clim_h2o2 (clim_t ∗clim, double t, double lat, double p)
  Climatology of H2O2 number concentrations.
- void clim_h2o2_init (ctl_t ∗ctl, clim_t ∗clim)
  Initialization function for H2O2 climatology.
- double clim_tropo (clim_t ∗clim, double t, double lat)
  Climatology of tropopause pressure.
- void clim_tropo_init (clim_t ∗clim)
  Initialize tropopause climatology.
- void compress_pack (char ∗varname, float ∗array, size_t nxy, size_t nz, int decompress, FILE ∗inout)
  Pack or unpack array.
- void day2doy (int year, int mon, int day, int ∗doy)
  Compress or decompress array with zfp.
- 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, clim_t ∗clim, double t, met_t ∗∗met0, met_t ∗∗met1)
  Get meteo data for given time step.
- void get_met_help (ctl_t ∗ctl, double t, int direct, char ∗metbase, double dt_met, char ∗filename)
  Get meteo data for time step.

- 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 meteo 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 meteo 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)

  Spatial interpolation of meteo 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 meteo data.
- void jsec2time (double jsec, int ∗year, int ∗mon, int ∗day, int ∗hour, int ∗min, int ∗sec, double ∗remain)

  Temporal interpolation of meteo data.
- double lapse_rate (double t, double h2o)

  Calculate moist adiabatic lapse rate.
- 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.
- double nat_temperature (double p, double h2o, double hno3)

  Calculate NAT existence temperature.
- void quicksort (double arr[ ], int brr[ ], int low, int high)

  Parallel quicksort.
- int quicksort_partition (double arr[ ], int brr[ ], int low, int high)

  Partition function for quicksort.
- int read_atm (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)

  Read atmospheric data.
- int read_atm_asc (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)

  Read atmospheric data in ASCII format.
- int read_atm_bin (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)

  Read atmospheric data in binary format.
- int read_atm_clams (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)

  Read atmospheric data in CLaMS format.
- int read_atm_nc (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)

  Read atmospheric data in netCDF format.
- void read_clim (ctl_t ∗ctl, clim_t ∗clim)

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

  Read control parameters.
- int read_met (char ∗filename, ctl_t ∗ctl, clim_t ∗clim, met_t ∗met)

  Read meteo data file.
- void read_met_bin_2d (FILE ∗in, met_t ∗met, float var[EX][EY], char ∗varname)

  Read 2-D meteo variable.
- void read_met_bin_3d (FILE ∗in, ctl_t ∗ctl, met_t ∗met, float var[EX][EY][EP], char ∗varname, int precision, double tolerance)

  Read 3-D meteo variable.
- void read_met_cape (clim_t ∗clim, met_t ∗met)

  Calculate convective available potential energy.

- void read_met_cloud (ctl_t ∗ctl, met_t ∗met)

    Calculate cloud properties.
- void read_met_detrend (ctl_t ∗ctl, met_t ∗met)

    Apply detrending method to temperature and winds.
- void read_met_extrapolate (met_t ∗met)

    Extrapolate meteo data at lower boundary.
- void read_met_geopot (ctl_t ∗ctl, met_t ∗met)

    Calculate geopotential heights.
- void read_met_grid (char ∗filename, int ncid, ctl_t ∗ctl, met_t ∗met)

    Read coordinates of meteo data.
- void read_met_levels (int ncid, ctl_t ∗ctl, met_t ∗met)

    Read meteo data on vertical levels.
- void read_met_ml2pl (ctl_t ∗ctl, met_t ∗met, float var[EX][EY][EP])

    Convert meteo data from model levels to pressure levels.
- int read_met_nc_2d (int ncid, char ∗varname, char ∗varname2, ctl_t ∗ctl, met_t ∗met, float dest[EX][EY], float scl, int init)

    Read and convert 2D variable from meteo data file.
- int read_met_nc_3d (int ncid, char ∗varname, char ∗varname2, ctl_t ∗ctl, met_t ∗met, float dest[EX][EY][EP], float scl, int init)

    Read and convert 3D variable from meteo data file.
- void read_met_pbl (met_t ∗met)

    Calculate pressure of the boundary layer.
- void read_met_periodic (met_t ∗met)

    Create meteo 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 meteo data.
- void read_met_surface (int ncid, met_t ∗met, ctl_t ∗ctl)

    Read surface data.
- void read_met_tropo (ctl_t ∗ctl, clim_t ∗clim, met_t ∗met)

    Calculate tropopause data.
- void read_obs (char ∗filename, double ∗rt, double ∗rz, double ∗rlon, double ∗rlat, double ∗robs, int ∗nobs)

    Read observation data.
- 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.
- double sedi (double p, double T, double rp, double rhop)

    Calculate sedimentation velocity.
- void spline (double ∗x, double ∗y, int n, double ∗x2, double ∗y2, int n2, int method)

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

    Calculate standard deviation.
- double sza (double sec, double lon, double lat)

    Calculate solar zenith angle.
- 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, const char ∗group, int output)

    Measure wall-clock time.
- double tropo_weight (clim_t ∗clim, double t, double lat, double p)

    Get weighting factor based on tropopause distance.

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

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

  Write atmospheric data in ASCII format.
- void write_atm_bin (const char *filename, ctl_t *ctl, atm_t *atm)

  Write atmospheric data in binary format.
- void write_atm_clams (ctl_t *ctl, atm_t *atm, double t)

  Write atmospheric data in CLaMS format.
- void write_atm_nc (const char *filename, ctl_t *ctl, atm_t *atm)

  Write atmospheric data in netCDF format.
- 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_grid_asc (const char *filename, ctl_t *ctl, double *cd, double *vmr_expl, double *vmr_impl, double t, double *z, double *lon, double *lat, double *area, double dz, int *np)

  Write gridded data in ASCII format.
- void write_grid_nc (const char *filename, ctl_t *ctl, double *cd, double *vmr_expl, double *vmr_impl, double t, double *z, double *lon, double *lat, double *area, double dz, int *np)

  Write gridded data in netCDF format.
- int write_met (char *filename, ctl_t *ctl, met_t *met)

  Read meteo data file.
- void write_met_bin_2d (FILE *out, met_t *met, float var[EX][EY], char *varname)

  Write 2-D meteo variable.
- void write_met_bin_3d (FILE *out, ctl_t *ctl, met_t *met, float var[EX][EY][EP], char *varname, int precision, double tolerance)

  Write 3-D meteo variable.
- 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_sample (const char *filename, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)

  Write sample 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 definitions.

Definition in file libtrac.c.

5.21.2 Function Documentation

5.21.2.1 buoyancy_frequency()

```
double buoyancy_frequency (
        double p0,
        double t0,
        double p1,
        double t1 )
```

Calculate buoyancy frequency.

Definition at line 29 of file libtrac.c.

```
                {

  double theta0 = THETA(p0, t0);
  double theta1 = THETA(p1, t1);
  double dz = RI / MA / G0 * 0.5 * (t0 + t1) * (log(p0) - log(p1));

  return sqrt(2. * G0 / (theta0 + theta1) * (theta1 - theta0) / dz);
}
```

5.21.2.2 cart2geo()

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

Convert Cartesian coordinates to geolocation.

Definition at line 44 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;
}
```

5.21.2.3 clim_hno3()

```
double clim_hno3 (
        clim_t * clim,
        double t,
        double lat,
        double p )
```

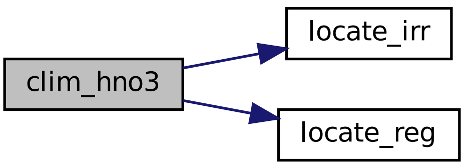
Climatology of HNO3 volume mixing ratios.

Definition at line 58 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_p[0])
    p = clim->hno3_p[0];
  else if (p > clim->hno3_p[clim->hno3_np - 1])
    p = clim->hno3_p[clim->hno3_np - 1];

  /* Check latitude... */
  if (lat < clim->hno3_lat[0])
    lat = clim->hno3_lat[0];
  else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
    lat = clim->hno3_lat[clim->hno3_nlat - 1];

  /* Get indices... */
```

```
  int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
  int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
  int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);

  /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
  double aux00 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec][ilat][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec][ilat + 1][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec + 1][ilat][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec + 1][ilat + 1][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim->hno3_lat[ilat], aux00,
              clim->hno3_lat[ilat + 1], aux01, lat);
  aux11 = LIN(clim->hno3_lat[ilat], aux10,
              clim->hno3_lat[ilat + 1], aux11, lat);
  aux00 = LIN(clim->hno3_time[isec], aux00,
              clim->hno3_time[isec + 1], aux11, sec);

  /* Convert from ppb to ppv... */
  return GSL_MAX(1e-9 * aux00, 0.0);
}
```

Here is the call graph for this function:



5.21.2.4 clim_hno3_init() `void clim_hno3_init ( clim_t * clim )`

Initialization function for HNO3 climatology.

Definition at line 116 of file libtrac.c.

```
                       {

  /* Write info... */
  LOG(1, "Initialize HNO3 data...");

  clim->hno3_ntime = 12;
  double hno3_time[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
  };
  memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));

  clim->hno3_nlat = 18;
  double hno3_lat[18] = {
```

```
    -85, -75, -65, -55, -45, -35, -25, -15, -5,
    5, 15, 25, 35, 45, 55, 65, 75, 85
  };
  memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));

  clim->hno3_np = 10;
  double hno3_p[10] = {
    4.64159, 6.81292, 10, 14.678, 21.5443,
    31.6228, 46.4159, 68.1292, 100, 146.78
  };
  memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));

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

```
      {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
      {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
      {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
      {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
      {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
      {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
      {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
      {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
      {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
      {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
      {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
      {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
      {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
      {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
      {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
      {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
     {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
      {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
      {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
      {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
      {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
      {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
      {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
      {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
      {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
      {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
      {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
      {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
      {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
      {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
      {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
      {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
      {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
      {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
     {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
      {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
      {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
      {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
      {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
      {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
      {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
      {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
      {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
      {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
      {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
      {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
      {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
      {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
      {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
      {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
      {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
      {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
     {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
      {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
      {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
      {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
      {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
      {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
      {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
      {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
      {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
      {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
      {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
      {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
      {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
      {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
      {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
      {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
      {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
      {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
      {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
      {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
      {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
      {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
      {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
      {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
      {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
      {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
      {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
      {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
      {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
      {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
      {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
      {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
      {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
      {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
```

```
      {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
      {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
      {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
      {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
      {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
      {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
      {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
      {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
      {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
      {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
      {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
      {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
      {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
      {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
      {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
      {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
      {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
      {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
      {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
      {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
      {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
      {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
      {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
      {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
      {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
      {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
      {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
      {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
      {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
      {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
      {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
      {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
      {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
      {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
      {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
      {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
      {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
      {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
      {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
      {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
      {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
      {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
      {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
      {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
      {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
      {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
      {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
      {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
      {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
      {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
      {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
      {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
  };
  memcpy(clim->hno3, hno3, sizeof(clim->hno3));

  /* Get range... */
  double hno3min = 1e99, hno3max = -1e99;
  for (int it = 0; it < clim->hno3_ntime; it++)
    for (int iz = 0; iz < clim->hno3_np; iz++)
      for (int iy = 0; iy < clim->hno3_nlat; iy++) {
        hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
        hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
      }

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->hno3_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->hno3_time[0], clim->hno3_time[1],
      clim->hno3_time[clim->hno3_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->hno3_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
      Z(clim->hno3_p[clim->hno3_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
      clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->hno3_lat[0], clim->hno3_lat[1],
      clim->hno3_lat[clim->hno3_nlat - 1]);
  LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
      1e-9 * hno3max);
}
```

5.21.2.5 clim_oh() double clim_oh (

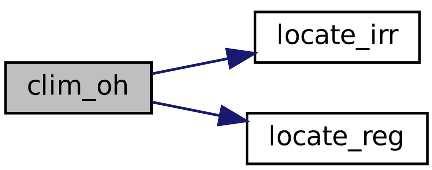
```
clim_t * clim,
double t,
double lat,
double p )
```

Climatology of OH number concentrations.

Definition at line 395 of file libtrac.c.

```
00399             {
00400
00401   /* Get seconds since begin of year... */
00402   double sec = FMOD(t, 365.25 * 86400.);
00403   while (sec < 0)
00404     sec += 365.25 * 86400.;
00405
00406   /* Check pressure... */
00407   if (p < clim->oh_p[clim->oh_np - 1])
00408     p = clim->oh_p[clim->oh_np - 1];
00409   else if (p > clim->oh_p[0])
00410     p = clim->oh_p[0];
00411
00412   /* Check latitude... */
00413   if (lat < clim->oh_lat[0])
00414     lat = clim->oh_lat[0];
00415   else if (lat > clim->oh_lat[clim->oh_nlat - 1])
00416     lat = clim->oh_lat[clim->oh_nlat - 1];
00417
00418   /* Get indices... */
00419   int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
00420   int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
00421   int ip = locate_irr(clim->oh_p, clim->oh_np, p);
00422
00423   /* Interpolate OH climatology... */
00424   double aux00 = LIN(clim->oh_p[ip],
00425                      clim->oh[isec][ip][ilat],
00426                      clim->oh_p[ip + 1],
00427                      clim->oh[isec][ip + 1][ilat], p);
00428   double aux01 = LIN(clim->oh_p[ip],
00429                      clim->oh[isec][ip][ilat + 1],
00430                      clim->oh_p[ip + 1],
00431                      clim->oh[isec][ip + 1][ilat + 1], p);
00432   double aux10 = LIN(clim->oh_p[ip],
00433                      clim->oh[isec + 1][ip][ilat],
00434                      clim->oh_p[ip + 1],
00435                      clim->oh[isec + 1][ip + 1][ilat], p);
00436   double aux11 = LIN(clim->oh_p[ip],
00437                      clim->oh[isec + 1][ip][ilat + 1],
00438                      clim->oh_p[ip + 1],
00439                      clim->oh[isec + 1][ip + 1][ilat + 1], p);
00440   aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
00441   aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
00442   aux00 =
00443     LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);
00444
00445   return GSL_MAX(aux00, 0.0);
00446 }
```

Here is the call graph for this function:



5.21.2.6 clim_oh_diurnal()

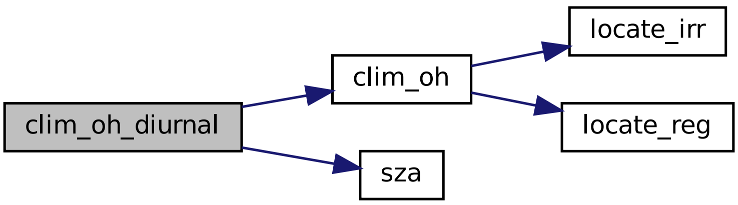
```
double clim_oh_diurnal (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    double p,
    double lon,
    double lat )
```

Climatology of OH number concentrations with diurnal variation.

Definition at line 450 of file libtrac.c.

```
                    {

  double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);

  if (sza2 <= M_PI / 2. * 89. / 90.)
    return oh * exp(-ctl->oh_chem_beta / cos(sza2));
  else
    return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
}
```

Here is the call graph for this function:



5.21.2.7 clim_oh_init()

```
void clim_oh_init (
    ctl_t * ctl,
    clim_t * clim )
```

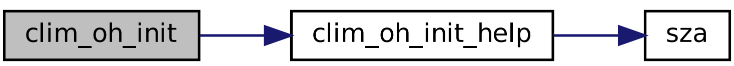
Initialization function for OH climatology.

Definition at line 468 of file libtrac.c.

```
                        {

  int nt, ncid, varid;

  double *help, ohmin = 1e99, ohmax = -1e99;

  /* Write info... */
  LOG(1, "Read OH data: %s", ctl->clim_oh_filename);

  /* Open netCDF file... */
  if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
    WARN("OH climatology data are missing!");
    return;
  }

  /* Read pressure data... */
  NC_INQ_DIM("press", &clim->oh_np, 2, CP);
  NC_GET_DOUBLE("press", clim->oh_p, 1);

  /* Check ordering of pressure data... */
```

```
  if (clim->oh_p[0] < clim->oh_p[1])
    ERRMSG("Pressure data are not descending!");

  /* Read latitudes... */
  NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
  NC_GET_DOUBLE("lat", clim->oh_lat, 1);

  /* Check ordering of latitudes... */
  if (clim->oh_lat[0] > clim->oh_lat[1])
    ERRMSG("Latitude data are not ascending!");

  /* Set time data for monthly means... */
  clim->oh_ntime = 12;
  clim->oh_time[0] = 1209600.00;
  clim->oh_time[1] = 3888000.00;
  clim->oh_time[2] = 6393600.00;
  clim->oh_time[3] = 9072000.00;
  clim->oh_time[4] = 11664000.00;
  clim->oh_time[5] = 14342400.00;
  clim->oh_time[6] = 16934400.00;
  clim->oh_time[7] = 19612800.00;
  clim->oh_time[8] = 22291200.00;
  clim->oh_time[9] = 24883200.00;
  clim->oh_time[10] = 27561600.00;
  clim->oh_time[11] = 30153600.00;

  /* Check number of timesteps... */
  NC_INQ_DIM("time", &nt, 12, 12);

  /* Read OH data... */
  ALLOC(help, double,
        clim->oh_nlat * clim->oh_np * clim->oh_ntime);
  NC_GET_DOUBLE("OH", help, 1);
  for (int it = 0; it < clim->oh_ntime; it++)
    for (int iz = 0; iz < clim->oh_np; iz++)
      for (int iy = 0; iy < clim->oh_nlat; iy++) {
        clim->oh[it][iz][iy] =
          help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
          / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
                              clim->oh_lat[iy]);
        ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
        ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
      }
  free(help);

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

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->oh_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->oh_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa",
      clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->oh_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
  LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
}
```

Here is the call graph for this function:



5.21.2.8 clim_oh_init_help()

```
double clim_oh_init_help (
    double beta,
    double time,
    double lat )
```

Apply diurnal correction to OH climatology.

Definition at line 555 of file libtrac.c.

```
{

  double aux, lon, sum = 0;

  int n = 0;

  /* Integrate day/night correction factor over longitude... */
  for (lon = -180; lon < 180; lon += 1) {
    aux = sza(time, lon, lat);
    if (aux <= M_PI / 2. * 85. / 90.)
      sum += exp(-beta / cos(aux));
    else
      sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
    n++;
  }
  return sum / (double) n;
}
```

Here is the call graph for this function:

5.21.2.9 clim_h2o2()

```
double clim_h2o2 (
    clim_t * clim,
    double t,
    double lat,
    double p )
```

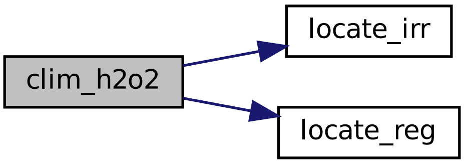
Climatology of H2O2 number concentrations.

Definition at line 578 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->h2o2_p[clim->h2o2_np - 1])
    p = clim->h2o2_p[clim->h2o2_np - 1];
  else if (p > clim->h2o2_p[0])
    p = clim->h2o2_p[0];

  /* Check latitude... */
  if (lat < clim->h2o2_lat[0])
    lat = clim->h2o2_lat[0];
  else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
    lat = clim->h2o2_lat[clim->h2o2_nlat - 1];

```

```
  /* Get indices... */
  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);

  /* Interpolate H2O2 climatology... */
  double aux00 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec][ip][ilat],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec][ip + 1][ilat], p);
  double aux01 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec][ip][ilat + 1],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec][ip + 1][ilat + 1], p);
  double aux10 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec + 1][ip][ilat],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec + 1][ip + 1][ilat], p);
  double aux11 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec + 1][ip][ilat + 1],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
  aux00 =
    LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
  aux11 =
    LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
  aux00 =
    LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);

  return GSL_MAX(aux00, 0.0);
}
```

Here is the call graph for this function:



5.21.2.10 clim_h2o2_init() `void clim_h2o2_init ( ctl_t * ctl, clim_t * clim )`

Initialization function for H2O2 climatology.

Definition at line 635 of file libtrac.c.

```
                  {

  int ncid, varid, it, iy, iz, nt;

  double *help, h2o2min = 1e99, h2o2max = -1e99;

  /* Write info... */
  LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);

  /* Open netCDF file... */
  if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
    WARN("H2O2 climatology data are missing!");
    return;
  }

```

```
  /* Read pressure data... */
  NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
  NC_GET_DOUBLE("press", clim->h2o2_p, 1);

  /* Check ordering of pressure data... */
  if (clim->h2o2_p[0] < clim->h2o2_p[1])
    ERRMSG("Pressure data are not descending!");

  /* Read latitudes... */
  NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
  NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);

  /* Check ordering of latitude data... */
  if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
    ERRMSG("Latitude data are not ascending!");

  /* Set time data (for monthly means)... */
  clim->h2o2_ntime = 12;
  clim->h2o2_time[0] = 1209600.00;
  clim->h2o2_time[1] = 3888000.00;
  clim->h2o2_time[2] = 6393600.00;
  clim->h2o2_time[3] = 9072000.00;
  clim->h2o2_time[4] = 11664000.00;
  clim->h2o2_time[5] = 14342400.00;
  clim->h2o2_time[6] = 16934400.00;
  clim->h2o2_time[7] = 19612800.00;
  clim->h2o2_time[8] = 22291200.00;
  clim->h2o2_time[9] = 24883200.00;
  clim->h2o2_time[10] = 27561600.00;
  clim->h2o2_time[11] = 30153600.00;

  /* Check number of timesteps... */
  NC_INQ_DIM("time", &nt, 12, 12);

  /* Read data... */
  ALLOC(help, double,
        clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
  NC_GET_DOUBLE("h2o2", help, 1);
  for (it = 0; it < clim->h2o2_ntime; it++)
    for (iz = 0; iz < clim->h2o2_np; iz++)
      for (iy = 0; iy < clim->h2o2_nlat; iy++) {
        clim->h2o2[it][iz][iy] =
          help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
        h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
        h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
      }
  free(help);

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

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->h2o2_time[0], clim->h2o2_time[1],
      clim->h2o2_time[clim->h2o2_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
      Z(clim->h2o2_p[clim->h2o2_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
      clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->h2o2_lat[0], clim->h2o2_lat[1],
      clim->h2o2_lat[clim->h2o2_nlat - 1]);
  LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
}
```

5.21.2.11 clim_tropo()

```
double clim_tropo (
    clim_t * clim,
    double t,
    double lat )
```

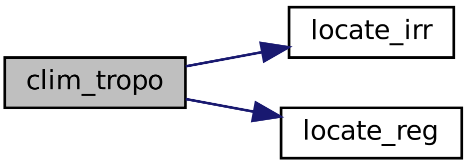
Climatology of tropopause pressure.

Definition at line 723 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_time, clim->tropo_ntime, sec);
  int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);

  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
  double p0 = LIN(clim->tropo_lat[ilat],
                  clim->tropo[isec][ilat],
                  clim->tropo_lat[ilat + 1],
                  clim->tropo[isec][ilat + 1], lat);
  double p1 = LIN(clim->tropo_lat[ilat],
                  clim->tropo[isec + 1][ilat],
                  clim->tropo_lat[ilat + 1],
                  clim->tropo[isec + 1][ilat + 1], lat);
  return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
}
```

Here is the call graph for this function:



5.21.2.12 clim_tropo_init() `void clim_tropo_init ( clim_t * clim )`

Initialize tropopause climatology.

Definition at line 751 of file libtrac.c.

```
                    {

  /* Write info... */
  LOG(1, "Initialize tropopause data...");

  clim->tropo_ntime = 12;
  double tropo_time[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
  };
  memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));

  clim->tropo_nlat = 73;
  double tropo_lat[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
  };
  memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));

```

```
  double tropo[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}
  };
  memcpy(clim->tropo, tropo, sizeof(clim->tropo));

  /* Get range... */
  double tropomin = 1e99, tropomax = -1e99;
  for (int it = 0; it < clim->tropo_ntime; it++)
    for (int iy = 0; iy < clim->tropo_nlat; iy++) {
      tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
      tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
    }

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->tropo_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->tropo_time[0], clim->tropo_time[1],
      clim->tropo_time[clim->tropo_ntime - 1]);
  LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->tropo_lat[0], clim->tropo_lat[1],
      clim->tropo_lat[clim->tropo_nlat - 1]);
  LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
      Z(tropomin));
  LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
}
```

5.21.2.13 compress_pack()

```
void compress_pack (
    char * varname,
    float * array,
    size_t nxy,
    size_t nz,
    int decompress,
    FILE * inout )
```

Pack or unpack array.

Definition at line 903 of file libtrac.c.

```
                 {

  double min[EP], max[EP], off[EP], scl[EP];

  unsigned short *sarray;

  /* Allocate... */
  ALLOC(sarray, unsigned short,
        nxy * nz);

  /* Read compressed stream and decompress array... */
  if (decompress) {

    /* Write info... */
    LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
        varname, 100. * sizeof(unsigned short) / sizeof(float));

    /* Read data... */
    FREAD(&scl, double,
          nz,
          inout);
    FREAD(&off, double,
          nz,
          inout);
    FREAD(sarray, unsigned short,
          nxy * nz,
          inout);

    /* Convert to float... */
```

```
#pragma omp parallel for default(shared)
    for (size_t ixy = 0; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++)
        array[ixy * nz + iz]
          = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
  }

  /* Compress array and output compressed stream... */
  else {

    /* Write info... */
    LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
        varname, 100. * sizeof(unsigned short) / sizeof(float));

    /* Get range... */
    for (size_t iz = 0; iz < nz; iz++) {
      min[iz] = array[iz];
      max[iz] = array[iz];
    }
    for (size_t ixy = 1; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++) {
        if (array[ixy * nz + iz] < min[iz])
          min[iz] = array[ixy * nz + iz];
        if (array[ixy * nz + iz] > max[iz])
          max[iz] = array[ixy * nz + iz];
      }

    /* Get offset and scaling factor... */
    for (size_t iz = 0; iz < nz; iz++) {
      scl[iz] = (max[iz] - min[iz]) / 65533.;
      off[iz] = min[iz];
    }

    /* Convert to short... */
#pragma omp parallel for default(shared)
    for (size_t ixy = 0; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++)
        if (scl[iz] != 0)
          sarray[ixy * nz + iz] = (unsigned short)
            ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
        else
          sarray[ixy * nz + iz] = 0;

    /* Write data... */
    FWRITE(&scl, double,
           nz,
           inout);
    FWRITE(&off, double,
           nz,
           inout);
    FWRITE(sarray, unsigned short,
           nxy * nz,
           inout);
  }

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

5.21.2.14 day2doy()

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

Compress or decompress array with zfp.

Compress or decompress array with zstd.

Get day of year from date.

Definition at line 1144 of file libtrac.c.

```
            {

  const int
    d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
```

```
01152     d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01153
01154   /* Get day of year... */
01155   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01156     *doy = d01[mon - 1] + day - 1;
01157   else
01158     *doy = d0[mon - 1] + day - 1;
01159 }
```

5.21.2.15 doy2day()

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

Get date from day of year.

Definition at line 1163 of file libtrac.c.

```
01167                 {
01168
01169   const int
01170     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01171     d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01172
01173   int i;
01174
01175   /* Get month and day... */
01176   if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01177     for (i = 11; i > 0; i--)
01178       if (d01[i] <= doy)
01179         break;
01180     *mon = i + 1;
01181     *day = doy - d01[i] + 1;
01182   } else {
01183     for (i = 11; i > 0; i--)
01184       if (d0[i] <= doy)
01185         break;
01186     *mon = i + 1;
01187     *day = doy - d0[i] + 1;
01188   }
01189 }
```

5.21.2.16 geo2cart()

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

Convert geolocation to Cartesian coordinates.

Definition at line 1193 of file libtrac.c.

```
01197                  {
01198
01199   double radius = z + RE;
01200   x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01201   x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01202   x[2] = radius * sin(lat / 180. * M_PI);
01203 }
```

5.21.2.17 get_met()

```
void get_met (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    met_t ** met0,
    met_t ** met1 )
```

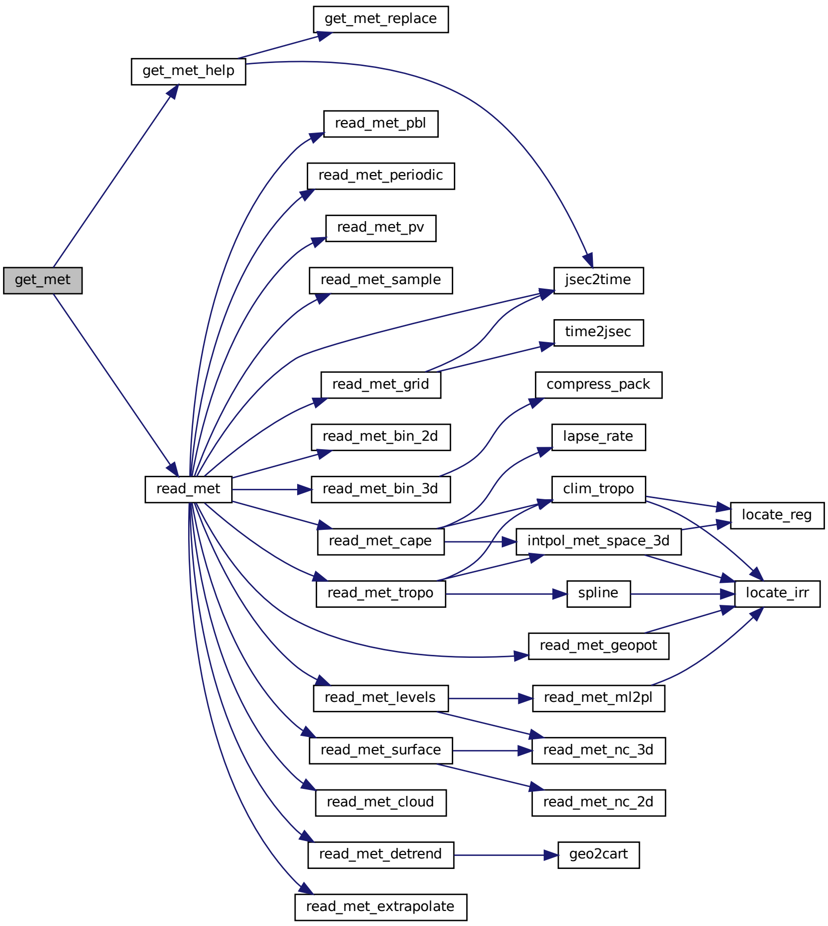
Get meteo data for given time step.

Definition at line 1207 of file libtrac.c.

```
                  {

  static int init;

  met_t *mets;

  char cachefile[LEN], cmd[2 * LEN], filename[LEN];

  /* Set timer... */
  SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);

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

    /* Read meteo data... */
    get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
                 ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met0))
      ERRMSG("Cannot open file!");

    get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
                 ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met1))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met0up = *met0;
    met_t *met1up = *met1;
#ifdef ASYNCIO
#pragma acc update device(met0up[:1],met1up[:1]) async(5)
#else
#pragma acc update device(met0up[:1],met1up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
                   ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }

  /* Read new data for forward trajectories... */
  if (t > (*met1)->time) {

    /* Pointer swap... */
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;

    /* Read new meteo data... */
    get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met1))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met1up = *met1;
#ifdef ASYNCIO
#pragma acc update device(met1up[:1]) async(5)
#else
#pragma acc update device(met1up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
```

```
      get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
                   cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }
  /* Read new data for backward trajectories... */
  if (t < (*met0)->time) {

    /* Pointer swap... */
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;

    /* Read new meteo data... */
    get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met0))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met0up = *met0;
#ifdef ASYNCIO
#pragma acc update device(met0up[:1]) async(5)
#else
#pragma acc update device(met0up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
                   cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }
  /* Check that grids are consistent... */
  if ((*met0)->nx != 0 && (*met1)->nx != 0) {
    if ((*met0)->nx != (*met1)->nx
        || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
      ERRMSG("Meteo grid dimensions do not match!");
    for (int ix = 0; ix < (*met0)->nx; ix++)
      if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
        ERRMSG("Meteo grid longitudes do not match!");
    for (int iy = 0; iy < (*met0)->ny; iy++)
      if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
        ERRMSG("Meteo grid latitudes do not match!");
    for (int ip = 0; ip < (*met0)->np; ip++)
      if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
        ERRMSG("Meteo grid pressure levels do not match!");
  }
}
```

Here is the call graph for this function:



5.21.2.18 get_met_help()

```
void get_met_help (
    ctl_t * ctl,
    double t,
    int direct,
    char * metbase,
    double dt_met,
    char * filename )
```

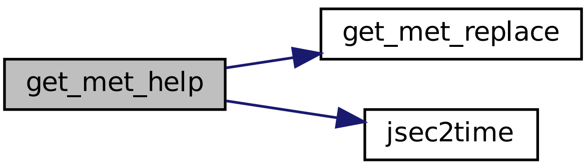
Get meteo data for time step.

Definition at line 1345 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 of MPTRAC meteo files... */
  if (ctl->clams_met_data == 0) {
    if (ctl->met_type == 0)
      sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
    else if (ctl->met_type == 1)
      sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
    else if (ctl->met_type == 2)
      sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
    else if (ctl->met_type == 3)
      sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
    else if (ctl->met_type == 4)
      sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", 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);
  }

  /* Set filename of CLaMS meteo files... */
  else {
    sprintf(filename, "%s_YYMMDDHH.nc", metbase);
    sprintf(repl, "%d", year);
    get_met_replace(filename, "YYYY", repl);
    sprintf(repl, "%d", year % 100);
    get_met_replace(filename, "YY", 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:



5.21.2.19 get_met_replace() `void get_met_replace ( char * orig, char * search, char * repl )`

Replace template strings in filename.

Definition at line 1408 of file libtrac.c.

```
                  {
```

```
  char buffer[LEN];

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

    /* Replace sub-string... */
    char *ch;
    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);
  }
}
```

5.21.2.20 intpol_met_space_3d()

```
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 )
```

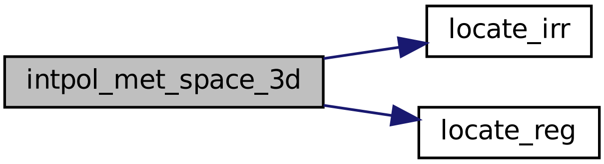
Spatial interpolation of meteo data.

Definition at line 1432 of file libtrac.c.

```
                 {

  /* Initialize interpolation... */
  if (init) {

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

    /* Get interpolation indices... */
    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);

    /* Get interpolation weights... */
    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:



5.21.2.21 intpol_met_space_2d()

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

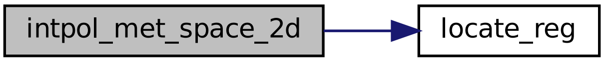
Spatial interpolation of meteo data.

Definition at line 1489 of file libtrac.c.

```
              {

  /* Initialize interpolation... */
  if (init) {

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

    /* Get interpolation indices... */
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);

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

```
          *var = aux11;
        else
          *var = aux01;
      } else {
        if (cw[1] < 0.5)
          *var = aux10;
        else
          *var = aux00;
      }
    }
}
```

Here is the call graph for this function:



5.21.2.22 intpol_met_time_3d()

```
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 )
```

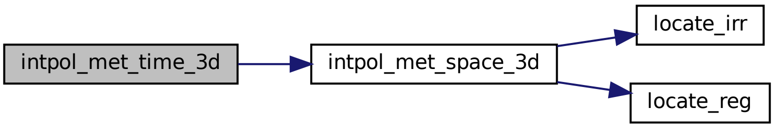
Spatial interpolation of meteo data.

Temporal interpolation of meteo data.

Definition at line 1649 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, 0);

  /* 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:



5.21.2.23 intpol_met_time_2d() void intpol_met_time_2d (

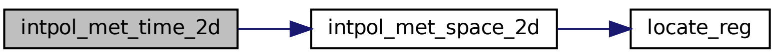
```
met_t * met0,
float array0[EX][EY],
met_t * met1,
float array1[EX][EY],
double ts,
double lon,
double lat,
double * var,
int * ci,
double * cw,
int init )
```

Temporal interpolation of meteo data.

Definition at line 1678 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, 0);

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

  /* Interpolate... */
  if (isfinite(var0) && isfinite(var1))
    *var = wt * (var0 - var1) + var1;
  else if (wt < 0.5)
    *var = var1;
  else
    *var = var0;
}
```

Here is the call graph for this function:



5.21.2.24 jsec2time()

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

Temporal interpolation of meteo data.

Convert seconds to date.

Definition at line 1742 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);
}
```

5.21.2.25 lapse_rate()

```
double lapse_rate (
    double t,
    double h2o )
```

Calculate moist adiabatic lapse rate.

Definition at line 1775 of file libtrac.c.

```
                   {

  /*
     Calculate moist adiabatic lapse rate [K/km] from temperature [K]
     and water vapor volume mixing ratio [1].

     Reference: https://en.wikipedia.org/wiki/Lapse_rate
   */

  const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));

  return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
}
```

5.21.2.26 locate_irr()

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

Find array index for irregular grid.

Definition at line 1793 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;
}
```

5.21.2.27 locate_reg()

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

Find array index for regular grid.

Definition at line 1823 of file libtrac.c.

```
                 {

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

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

5.21.2.28 nat_temperature()

```
double nat_temperature (
    double p,
    double h2o,
    double hno3 )
```

Calculate NAT existence temperature.

Definition at line 1842 of file libtrac.c.

```
                      {

  /* Check water vapor vmr... */
```

```
  h2o = GSL_MAX(h2o, 0.1e-6);

  /* Calculate T_NAT... */
  double p_hno3 = hno3 * p / 1.333224;
  double p_h2o = h2o * p / 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 tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
  if (x2 > 0)
    tnat = x2;

  return tnat;
}
```

5.21.2.29 quicksort()

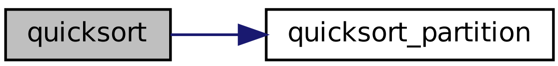
```
void quicksort (
    double arr[],
    int brr[],
    int low,
    int high )
```

Parallel quicksort.

Definition at line 1866 of file libtrac.c.

```
                {

  if (low < high) {
    int pi = quicksort_partition(arr, brr, low, high);

#pragma omp task firstprivate(arr,brr,low,pi)
    {
      quicksort(arr, brr, low, pi - 1);
    }

    // #pragma omp task firstprivate(arr,brr,high,pi)
    {
      quicksort(arr, brr, pi + 1, high);
    }
  }
}
```

Here is the call graph for this function:



5.21.2.30 quicksort_partition()

```
int quicksort_partition (
    double arr[],
    int brr[],
    int low,
    int high )
```

Partition function for quicksort.

Definition at line 1889 of file libtrac.c.

```
              {

  double pivot = arr[high];
  int i = (low - 1);

  for (int j = low; j <= high - 1; j++)
    if (arr[j] <= pivot) {
      i++;
      SWAP(arr[i], arr[j], double);
      SWAP(brr[i], brr[j], int);
    }
  SWAP(arr[high], arr[i + 1], double);
  SWAP(brr[high], brr[i + 1], int);

  return (i + 1);
}
```

5.21.2.31 read_atm()

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

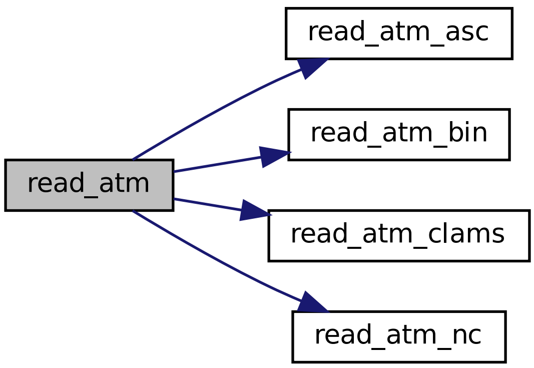
Read atmospheric data.

Definition at line 1912 of file libtrac.c.

```
                 {

  int result;

  /* Set timer... */
  SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);

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

  /* Write info... */
  LOG(1, "Read atmospheric data: %s", filename);

  /* Read ASCII data... */
  if (ctl->atm_type == 0)
    result = read_atm_asc(filename, ctl, atm);

  /* Read binary data... */
  else if (ctl->atm_type == 1)
    result = read_atm_bin(filename, ctl, atm);

  /* Read netCDF data... */
  else if (ctl->atm_type == 2)
    result = read_atm_nc(filename, ctl, atm);

  /* Read CLaMS data... */
  else if (ctl->atm_type == 3)
    result = read_atm_clams(filename, ctl, atm);

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

  /* Check result... */
  if (result != 1)
    return 0;

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of particles: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
```

```
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  for (int iq = 0; iq < ctl->nq; iq++) {
    char msg[LEN];
    sprintf(msg, "Quantity %s range: %s ... %s %s",
            ctl->qnt_name[iq], ctl->qnt_format[iq],
            ctl->qnt_format[iq], ctl->qnt_unit[iq]);
    gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
    LOG(2, msg, mini, maxi);
  }

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

Here is the call graph for this function:



5.21.2.32 read_atm_asc()

```
int read_atm_asc (
        const char * filename,
        ctl_t * ctl,
        atm_t * atm )
```

Read atmospheric data in ASCII format.

Definition at line 1983 of file libtrac.c.

```
                      {

  FILE *in;

  /* Open file... */
  if (!(in = fopen(filename, "r"))) {
    WARN("Cannot open file!");
    return 0;
  }

  /* Read line... */
  char line[LEN];
  while (fgets(line, LEN, in)) {

    /* Read data... */
    char *tok;
    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 (int 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);

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

5.21.2.33 read_atm_bin() `int read_atm_bin ( const char * filename, ctl_t * ctl, atm_t * atm )`

Read atmospheric data in binary format.

Definition at line 2026 of file libtrac.c.

```
                    {

  FILE *in;

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

  /* Check version of binary data... */
  int version;
  FREAD(&version, int,
        1,
        in);
  if (version != 100)
    ERRMSG("Wrong version of binary data!");

  /* 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 (int iq = 0; iq < ctl->nq; iq++)
    FREAD(atm->q[iq], double,
             (size_t) atm->np,
          in);

  /* Read final flag... */
  int final;
  FREAD(&final, int,
        1,
        in);
  if (final != 999)
    ERRMSG("Error while reading binary data!");

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

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

5.21.2.34 read_atm_clams()

```
int read_atm_clams (
        const char * filename,
        ctl_t * ctl,
        atm_t * atm )
```

Read atmospheric data in CLaMS format.

Definition at line 2083 of file libtrac.c.

```
                     {

  int ncid, varid;

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

  /* Get dimensions... */
  NC_INQ_DIM("NPARTS", &atm->np, 1, NP);

  /* Get time... */
  if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
    NC(nc_get_var_double(ncid, varid, atm->time));
  } else {
    WARN("TIME_INIT not found use time instead!");
    double time_init;
    NC_GET_DOUBLE("time", &time_init, 1);
    for (int ip = 0; ip < atm->np; ip++) {
      atm->time[ip] = time_init;
    }
  }

  /* Read zeta coordinate, pressure is optional... */
  if (ctl->vert_coord_ap == 1) {
    NC_GET_DOUBLE("ZETA", atm->zeta, 1);
    NC_GET_DOUBLE("PRESS", atm->p, 0);
  }

  /* Read pressure, zeta coordinate is optional... */
  else {
    NC_GET_DOUBLE("PRESS", atm->p, 1);
    NC_GET_DOUBLE("ZETA", atm->zeta, 0);
  }

  /* Read longitude and latitude... */
  NC_GET_DOUBLE("LON", atm->lon, 1);
  NC_GET_DOUBLE("LAT", atm->lat, 1);

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

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

5.21.2.35 read_atm_nc()

```
int read_atm_nc (
        const char * filename,
        ctl_t * ctl,
        atm_t * atm )
```

Read atmospheric data in netCDF format.

Definition at line 2134 of file libtrac.c.

```
                     {

  int ncid, varid;

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

  /* Get dimensions... */
  NC_INQ_DIM("obs", &atm->np, 1, NP);

  /* Read geolocations... */
  NC_GET_DOUBLE("time", atm->time, 1);
```

```
  NC_GET_DOUBLE("press", atm->p, 1);
  NC_GET_DOUBLE("lon", atm->lon, 1);
  NC_GET_DOUBLE("lat", atm->lat, 1);

  /* Read variables... */
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

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

5.21.2.36 read_clim()

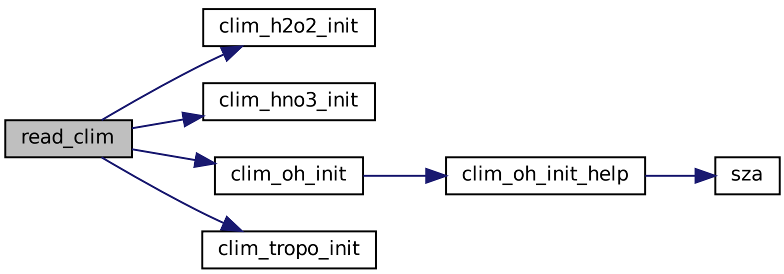
```
void read_clim (
    ctl_t * ctl,
    clim_t * clim )
```

Read climatological data.

Definition at line 2167 of file libtrac.c.

```
                 {

  /* Set timer... */
  SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);

  /* Init tropopause climatology... */
  clim_tropo_init(clim);

  /* Init HNO3 climatology... */
  clim_hno3_init(clim);

  /* Read OH climatology... */
  if (ctl->clim_oh_filename[0] != '-')
    clim_oh_init(ctl, clim);

  /* Read H2O2 climatology... */
  if (ctl->clim_h2o2_filename[0] != '-')
    clim_h2o2_init(ctl, clim);
}
```

Here is the call graph for this function:



5.21.2.37 read_ctl()

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

Read control parameters.

Definition at line 2191 of file libtrac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);

  /* Write info... */
  LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
      "(executable: %s | version: %s | compiled: %s, %s)\n",
      argv[0], VERSION, __DATE__, __TIME__);

  /* Initialize quantity indices... */
  ctl->qnt_idx = -1;
  ctl->qnt_ens = -1;
  ctl->qnt_stat = -1;
  ctl->qnt_m = -1;
  ctl->qnt_vmr = -1;
  ctl->qnt_rp = -1;
  ctl->qnt_rhop = -1;
  ctl->qnt_ps = -1;
  ctl->qnt_ts = -1;
  ctl->qnt_zs = -1;
  ctl->qnt_us = -1;
  ctl->qnt_vs = -1;
  ctl->qnt_pbl = -1;
  ctl->qnt_pt = -1;
  ctl->qnt_tt = -1;
  ctl->qnt_zt = -1;
  ctl->qnt_h2ot = -1;
  ctl->qnt_z = -1;
  ctl->qnt_p = -1;
  ctl->qnt_t = -1;
  ctl->qnt_rho = -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_pct = -1;
  ctl->qnt_pcb = -1;
  ctl->qnt_cl = -1;
  ctl->qnt_plcl = -1;
  ctl->qnt_plfc = -1;
  ctl->qnt_pel = -1;
  ctl->qnt_cape = -1;
  ctl->qnt_cin = -1;
  ctl->qnt_hno3 = -1;
  ctl->qnt_oh = -1;
  ctl->qnt_vmrimpl = -1;
  ctl->qnt_mloss_oh = -1;
  ctl->qnt_mloss_h2o2 = -1;
  ctl->qnt_mloss_wet = -1;
  ctl->qnt_mloss_dry = -1;
  ctl->qnt_mloss_decay = -1;
  ctl->qnt_psat = -1;
  ctl->qnt_psice = -1;
  ctl->qnt_pw = -1;
  ctl->qnt_sh = -1;
  ctl->qnt_rh = -1;
  ctl->qnt_rhice = -1;
  ctl->qnt_theta = -1;
  ctl->qnt_zeta = -1;
  ctl->qnt_tvirt = -1;
  ctl->qnt_lapse = -1;
  ctl->qnt_vh = -1;
  ctl->qnt_vz = -1;
  ctl->qnt_pv = -1;
  ctl->qnt_tdew = -1;
  ctl->qnt_tice = -1;
  ctl->qnt_tsts = -1;
  ctl->qnt_tnat = -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_LONGNAME", iq, ctl->qnt_name[iq],
             ctl->qnt_longname[iq]);
    scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
             ctl->qnt_format[iq]);

    /* Try to identify quantity... */
    SET_QNT(qnt_idx, "idx", "particle index", "-")
      SET_QNT(qnt_ens, "ens", "ensemble index", "-")
      SET_QNT(qnt_stat, "stat", "station flag", "-")
      SET_QNT(qnt_m, "m", "mass", "kg")
      SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
      SET_QNT(qnt_rp, "rp", "particle radius", "microns")
      SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
      SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
      SET_QNT(qnt_ts, "ts", "surface temperature", "K")
      SET_QNT(qnt_zs, "zs", "surface height", "km")
      SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
      SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
      SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
      SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
      SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
      SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
      SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
      SET_QNT(qnt_z, "z", "geopotential height", "km")
      SET_QNT(qnt_p, "p", "pressure", "hPa")
      SET_QNT(qnt_t, "t", "temperature", "K")
      SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
      SET_QNT(qnt_u, "u", "zonal wind", "m/s")
      SET_QNT(qnt_v, "v", "meridional wind", "m/s")
      SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
      SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
      SET_QNT(qnt_o3, "o3", "ozone", "ppv")
      SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
      SET_QNT(qnt_iwc, "iwc", "cloud liquid water content", "kg/kg")
      SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
      SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
      SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
      SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
      SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
      SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
      SET_QNT(qnt_cape, "cape", "convective available potential energy",
              "J/kg")
      SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
      SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
      SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
      SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
      SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
      SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
              "kg")
      SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
              "kg")
      SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
              "kg")
      SET_QNT(qnt_mloss_decay, "mloss_decay",
              "mass loss due to exponential decay", "kg")
      SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
      SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
      SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
      SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
      SET_QNT(qnt_rh, "rh", "relative humidity", "%%")
      SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%%")
      SET_QNT(qnt_theta, "theta", "potential temperature", "K")
      SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
      SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
      SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
      SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
      SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
      SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
      SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
      SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
      SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
      SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
      scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
  }

  /* netCDF I/O parameters... */
  ctl->chunkszhint =
    (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
                      NULL);
  ctl->read_mode =
```

```
    (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);

  /* Vertical coordinates and velocities... */
  ctl->vert_coord_ap =
    (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
  ctl->vert_coord_met =
    (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
  ctl->vert_vel =
    (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
  ctl->clams_met_data =
    (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);

  /* 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, "180", NULL);

  /* Meteo data... */
  scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
  ctl->met_type =
    (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
  ctl->met_nc_scale =
    (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", 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);
  if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
    ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
  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);
  if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
    ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
  ctl->met_detrend =
    scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", 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_geopot_sx
    = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
  ctl->met_geopot_sy
    = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
  ctl->met_relhum
    = (int) scan_ctl(filename, argc, argv, "MET_RELHUM", -1, "0", NULL);
  ctl->met_tropo =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
  if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
    ERRMSG("Set MET_TROPO = 0 ... 5!");
  ctl->met_tropo_lapse =
    scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
  ctl->met_tropo_nlev =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
  ctl->met_tropo_lapse_sep =
    scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
  ctl->met_tropo_nlev_sep =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
                   NULL);
  ctl->met_tropo_pv =
    scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
  ctl->met_tropo_theta =
    scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
  ctl->met_tropo_spline =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
  ctl->met_cloud =
    (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
  if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
    ERRMSG("Set MET_CLOUD = 0 ... 3!");
  ctl->met_cloud_min =
    scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
  ctl->met_dt_out =
    scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
  ctl->met_cache =
    (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);

  /* Sorting... */
  ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);

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

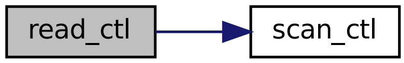
```
  /* Advection parameters... */
  ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
  if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
    ERRMSG("Set ADVECT to 1, 2, or 4!");
  ctl->reflect =
    (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);

  /* 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);

  /* Convection... */
  ctl->conv_cape
    = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
  ctl->conv_cin
    = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
  ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
  ctl->conv_mix
    = (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "1", NULL);
  ctl->conv_mix_bot
    = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
  ctl->conv_mix_top
    = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);

  /* Boundary conditions... */
  ctl->bound_mass =
    scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
  ctl->bound_mass_trend =
    scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
  ctl->bound_vmr =
    scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
  ctl->bound_vmr_trend =
    scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
  ctl->bound_lat0 =
    scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
  ctl->bound_lat1 =
    scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
  ctl->bound_p0 =
    scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
  ctl->bound_p1 =
    scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
  ctl->bound_dps =
    scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
  ctl->bound_dzs =
    scan_ctl(filename, argc, argv, "BOUND_DZS", -1, "-999", NULL);
  ctl->bound_zetas =
    scan_ctl(filename, argc, argv, "BOUND_ZETAS", -1, "-999", NULL);
  ctl->bound_pbl =
    (int) scan_ctl(filename, argc, argv, "BOUND_PBL", -1, "0", NULL);

  /* Species parameters... */
  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
  if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
    ctl->molmass = 120.907;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
  } else if (strcasecmp(ctl->species, "CFCl3") == 0) {
    ctl->molmass = 137.359;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
  } else if (strcasecmp(ctl->species, "CH4") == 0) {
    ctl->molmass = 16.043;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 2.45e-12;
    ctl->oh_chem[1] = 1775;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
  } else if (strcasecmp(ctl->species, "CO") == 0) {
    ctl->molmass = 28.01;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 6.9e-33;
    ctl->oh_chem[1] = 2.1;
    ctl->oh_chem[2] = 1.1e-12;
    ctl->oh_chem[3] = -1.3;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
```

```
  } else if (strcasecmp(ctl->species, "CO2") == 0) {
    ctl->molmass = 44.009;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
  } else if (strcasecmp(ctl->species, "H2O") == 0) {
    ctl->molmass = 18.01528;
  } else if (strcasecmp(ctl->species, "N2O") == 0) {
    ctl->molmass = 44.013;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
  } else if (strcasecmp(ctl->species, "NH3") == 0) {
    ctl->molmass = 17.031;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 1.7e-12;
    ctl->oh_chem[1] = 710;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
  } else if (strcasecmp(ctl->species, "HNO3") == 0) {
    ctl->molmass = 63.012;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
  } else if (strcasecmp(ctl->species, "NO") == 0) {
    ctl->molmass = 30.006;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 7.1e-31;
    ctl->oh_chem[1] = 2.6;
    ctl->oh_chem[2] = 3.6e-11;
    ctl->oh_chem[3] = 0.1;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
  } else if (strcasecmp(ctl->species, "NO2") == 0) {
    ctl->molmass = 46.005;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 1.8e-30;
    ctl->oh_chem[1] = 3.0;
    ctl->oh_chem[2] = 2.8e-11;
    ctl->oh_chem[3] = 0.0;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
  } else if (strcasecmp(ctl->species, "O3") == 0) {
    ctl->molmass = 47.997;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 1.7e-12;
    ctl->oh_chem[1] = 940;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
  } else if (strcasecmp(ctl->species, "SF6") == 0) {
    ctl->molmass = 146.048;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
  } else if (strcasecmp(ctl->species, "SO2") == 0) {
    ctl->molmass = 64.066;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 2.9e-31;
    ctl->oh_chem[1] = 4.1;
    ctl->oh_chem[2] = 1.7e-12;
    ctl->oh_chem[3] = -0.2;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
  } else {
    ctl->molmass =
      scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
    ctl->oh_chem_reaction =
      (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
    ctl->h2o2_chem_reaction =
      (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
                     NULL);
    for (int ip = 0; ip < 4; ip++)
      ctl->oh_chem[ip] =
        scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
    ctl->dry_depo_vdep =
      scan_ctl(filename, argc, argv, "DRY_DEPO_VDEP", -1, "0", NULL);
    ctl->dry_depo_dp =
      scan_ctl(filename, argc, argv, "DRY_DEPO_DP", -1, "30", NULL);
    ctl->wet_depo_ic_a =
      scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
    ctl->wet_depo_ic_b =
      scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
    ctl->wet_depo_bc_a =
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
    ctl->wet_depo_bc_b =
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
    for (int ip = 0; ip < 3; ip++)
      ctl->wet_depo_ic_h[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
    for (int ip = 0; ip < 1; ip++)
      ctl->wet_depo_bc_h[ip] =
```

```
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
  }

  /* Wet deposition... */
  ctl->wet_depo_pre[0] =
    scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
  ctl->wet_depo_pre[1] =
    scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
  ctl->wet_depo_ic_ret_ratio =
    scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
  ctl->wet_depo_bc_ret_ratio =
    scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);

  /* OH chemistry... */
  ctl->oh_chem_beta =
    scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
           "../../data/clams_radical_species.nc", ctl->clim_oh_filename);

  /* H2O2 chemistry... */
  ctl->h2o2_chem_cc =
    scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
  scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
           "../../data/cams_H2O2.nc", ctl->clim_h2o2_filename);

  /* Chemistry grid... */
  ctl->chemgrid_z0 =
    scan_ctl(filename, argc, argv, "CHEMGRID_Z0", -1, "0", NULL);
  ctl->chemgrid_z1 =
    scan_ctl(filename, argc, argv, "CHEMGRID_Z1", -1, "100", NULL);
  ctl->chemgrid_nz =
    (int) scan_ctl(filename, argc, argv, "CHEMGRID_NZ", -1, "1", NULL);
  ctl->chemgrid_lon0 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LON0", -1, "-180", NULL);
  ctl->chemgrid_lon1 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LON1", -1, "180", NULL);
  ctl->chemgrid_nx =
    (int) scan_ctl(filename, argc, argv, "CHEMGRID_NX", -1, "360", NULL);
  ctl->chemgrid_lat0 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LAT0", -1, "-90", NULL);
  ctl->chemgrid_lat1 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LAT1", -1, "90", NULL);
  ctl->chemgrid_ny =
    (int) scan_ctl(filename, argc, argv, "CHEMGRID_NY", -1, "180", NULL);

  /* Exponential decay... */
  ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
  ctl->tdec_strat =
    scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);

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

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

  /* 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);
  ctl->grid_type =
    (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", 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 sample data... */
  scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
           ctl->sample_basename);
  scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
           ctl->sample_obsfile);
  ctl->sample_dx =
    scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
  ctl->sample_dz =
    scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", 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);
  ctl->stat_t0 =
    scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
  ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
}
```

Here is the call graph for this function:



5.21.2.38 read_met()

```
int read_met (
    char * filename,
    ctl_t * ctl,
    clim_t * clim,
    met_t * met )
```

Read meteo data file.

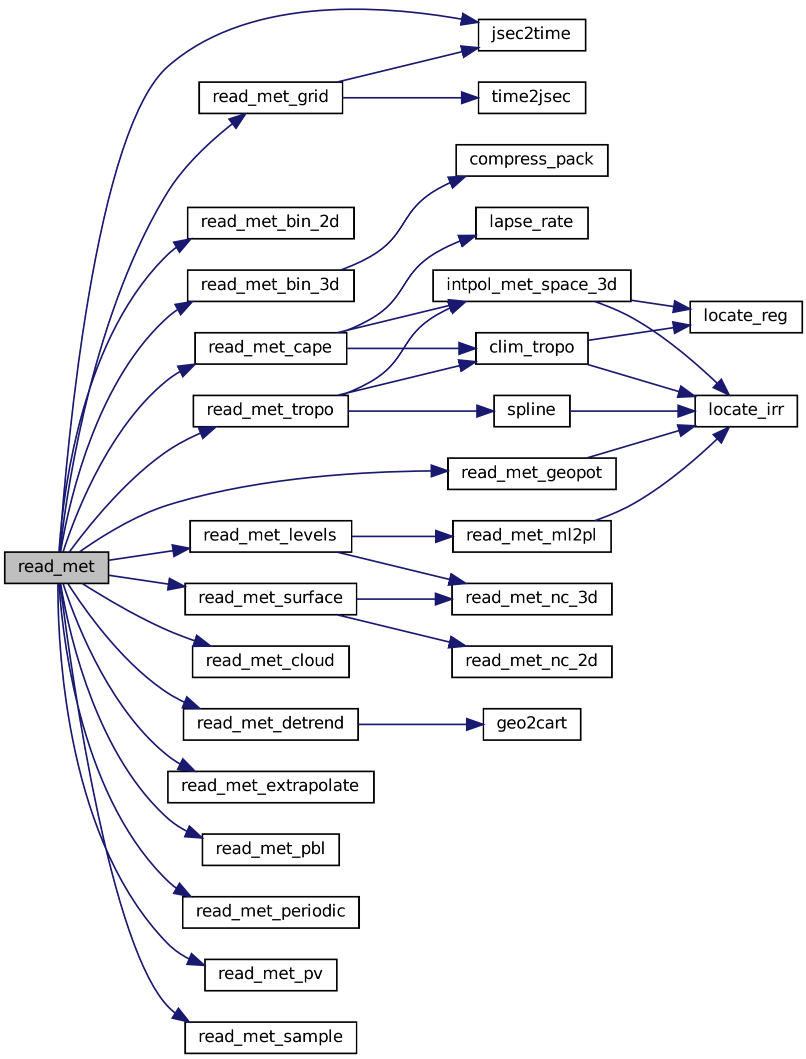
Definition at line 2782 of file libtrac.c.

```
                  {

  /* Write info... */
  LOG(1, "Read meteo data: %s", filename);

  /* Read netCDF data... */
  if (ctl->met_type == 0) {

    int ncid;

    /* Open netCDF file... */
    if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
        NC_NOERR) {
      WARN("Cannot open file!");
      return 0;
    }

    /* Read coordinates of meteo data... */
    read_met_grid(filename, ncid, ctl, met);

    /* Read meteo data on vertical levels... */
    read_met_levels(ncid, ctl, met);

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

    /* Read surface data... */
    read_met_surface(ncid, met, ctl);

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

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

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

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

    /* Calculate boundary layer data... */
    read_met_pbl(met);

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

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

```

    /* Calculate convective available potential energy... */
    read_met_cape(clim, met);

    /* Detrending... */
    read_met_detrend(ctl, met);

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

  /* Read binary data... */
  else if (ctl->met_type >= 1 && ctl->met_type <= 4) {

    FILE *in;

    double r;

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

    /* Set timer... */
    SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);

    /* Open file... */
    if (!(in = fopen(filename, "r"))) {
      WARN("Cannot open file!");
      return 0;
    }

    /* Check type of binary data... */
    int met_type;
    FREAD(&met_type, int,
          1,
          in);
    if (met_type != ctl->met_type)
      ERRMSG("Wrong MET_TYPE of binary data!");

    /* Check version of binary data... */
    int version;
    FREAD(&version, int,
          1,
          in);
    if (version != 100)
      ERRMSG("Wrong version of binary data!");

    /* Read time... */
    FREAD(&met->time, double,
          1,
          in);
    jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
    LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
        met->time, year, mon, day, hour, min);
    if (year < 1900 || year > 2100 || mon < 1 || mon > 12
        || day < 1 || day > 31 || hour < 0 || hour > 23)
      ERRMSG("Error while reading time!");

    /* Read dimensions... */
    FREAD(&met->nx, int,
          1,
          in);
    LOG(2, "Number of longitudes: %d", met->nx);
    if (met->nx < 2 || met->nx > EX)
      ERRMSG("Number of longitudes out of range!");

    FREAD(&met->ny, int,
          1,
          in);
    LOG(2, "Number of latitudes: %d", met->ny);
    if (met->ny < 2 || met->ny > EY)
      ERRMSG("Number of latitudes out of range!");

    FREAD(&met->np, int,
          1,
          in);
    LOG(2, "Number of levels: %d", met->np);
    if (met->np < 2 || met->np > EP)
      ERRMSG("Number of levels out of range!");

    /* Read grid... */
    FREAD(met->lon, double,
            (size_t) met->nx,
          in);
    LOG(2, "Longitudes: %g, %g ... %g deg",
        met->lon[0], met->lon[1], met->lon[met->nx - 1]);

    FREAD(met->lat, double,
            (size_t) met->ny,
```

```
        in);
    LOG(2, "Latitudes: %g, %g ... %g deg",
        met->lat[0], met->lat[1], met->lat[met->ny - 1]);

    FREAD(met->p, double,
          (size_t) met->np,
        in);
    LOG(2, "Altitude levels: %g, %g ... %g km",
        Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
    LOG(2, "Pressure levels: %g, %g ... %g hPa",
        met->p[0], met->p[1], met->p[met->np - 1]);

    /* Read surface data... */
    read_met_bin_2d(in, met, met->ps, "PS");
    read_met_bin_2d(in, met, met->ts, "TS");
    read_met_bin_2d(in, met, met->zs, "ZS");
    read_met_bin_2d(in, met, met->us, "US");
    read_met_bin_2d(in, met, met->vs, "VS");
    read_met_bin_2d(in, met, met->pbl, "PBL");
    read_met_bin_2d(in, met, met->pt, "PT");
    read_met_bin_2d(in, met, met->tt, "TT");
    read_met_bin_2d(in, met, met->zt, "ZT");
    read_met_bin_2d(in, met, met->h2ot, "H2OT");
    read_met_bin_2d(in, met, met->pct, "PCT");
    read_met_bin_2d(in, met, met->pcb, "PCB");
    read_met_bin_2d(in, met, met->cl, "CL");
    read_met_bin_2d(in, met, met->plcl, "PLCL");
    read_met_bin_2d(in, met, met->plfc, "PLFC");
    read_met_bin_2d(in, met, met->pel, "PEL");
    read_met_bin_2d(in, met, met->cape, "CAPE");
    read_met_bin_2d(in, met, met->cin, "CIN");

    /* Read level data... */
    read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
    read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
    read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
    read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
    read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
    read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
    read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
    read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
    read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
    read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);

    /* Read final flag... */
    int final;
    FREAD(&final, int,
        1,
        in);
    if (final != 999)
      ERRMSG("Error while reading binary data!");

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

  /* Not implemented... */
  else
    ERRMSG("MET_TYPE not implemented!");

  /* Copy wind data to cache... */
#ifdef UVW
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++) {
        met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
        met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
        met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
      }
#endif

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

Here is the call graph for this function:



5.21.2.39 read_met_bin_2d()

```
void read_met_bin_2d (
    FILE * in,
    met_t * met,
    float var[EX][EY],
    char * varname )
```

Read 2-D meteo variable.

Definition at line 3000 of file libtrac.c.

```
                           {

  float *help;

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

```
  /* Read uncompressed... */
  LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
  FREAD(help, float,
        (size_t) (met->nx * met->ny),
      in);

  /* Copy data... */
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];

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

5.21.2.40 read_met_bin_3d()

```
void read_met_bin_3d (
    FILE * in,
    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP],
    char * varname,
    int precision,
    double tolerance )
```

Read 3-D meteo variable.

Definition at line 3029 of file libtrac.c.

```
                  {

  float *help;

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

  /* Read uncompressed data... */
  if (ctl->met_type == 1) {
    LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
    FREAD(help, float,
          (size_t) (met->nx * met->ny * met->np),
        in);
  }

  /* Read packed data... */
  else if (ctl->met_type == 2)
    compress_pack(varname, help, (size_t) (met->ny * met->nx),
                  (size_t) met->np, 1, in);

  /* Read zfp data... */
  else if (ctl->met_type == 3) {
#ifdef ZFP
    compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
                 tolerance, 1, in);
#else
    ERRMSG("zfp compression not supported!");
    LOG(3, "%d %g", precision, tolerance);
#endif
  }

  /* Read zstd data... */
  else if (ctl->met_type == 4) {
#ifdef ZSTD
    compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
                  in);
#else
    ERRMSG("zstd compression not supported!");
#endif
  }

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];
```

```

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

Here is the call graph for this function:

5.21.2.41 read_met_cape() void read_met_cape (
clim_t * *clim*,
met_t * *met*)

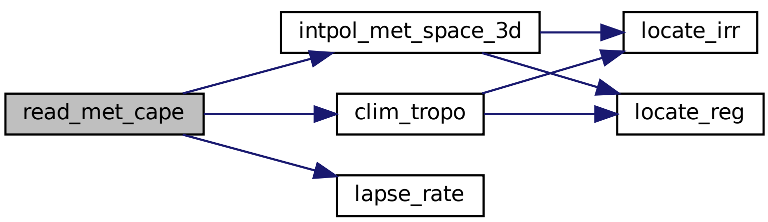
Calculate convective available potential energy.

Definition at line 3091 of file libtrac.c.

```
                 {

  /* Set timer... */
  SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
  LOG(2, "Calculate CAPE...");

  /* Vertical spacing (about 100 m)... */
  const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
      int n = 0;
      double h2o = 0, t, theta = 0;
      double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
      double ptop = pbot - 50.;
      for (int ip = 0; ip < met->np; ip++) {
        if (met->p[ip] <= pbot) {
          theta += THETA(met->p[ip], met->t[ix][iy][ip]);
          h2o += met->h2o[ix][iy][ip];
          n++;
        }
        if (met->p[ip] < ptop && n > 0)
          break;
      }
      theta /= n;
      h2o /= n;

      /* Cannot compute anything if water vapor is missing... */
      met->plcl[ix][iy] = GSL_NAN;
      met->plfc[ix][iy] = GSL_NAN;
      met->pel[ix][iy] = GSL_NAN;
      met->cape[ix][iy] = GSL_NAN;
      met->cin[ix][iy] = GSL_NAN;
      if (h2o <= 0)
        continue;

      /* Find lifted condensation level (LCL)... */
      ptop = P(20.);
      pbot = met->ps[ix][iy];
      do {
        met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
        t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
        if (RH(met->plcl[ix][iy], t, h2o) > 100.)
```

```
          ptop = met->plcl[ix][iy];
        else
          pbot = met->plcl[ix][iy];
      } while (pbot - ptop > 0.1);

      /* Calculate CIN up to LCL... */
      INTPOL_INIT;
      double dcape, dz, h2o_env, t_env;
      double p = met->ps[ix][iy];
      met->cape[ix][iy] = met->cin[ix][iy] = 0;
      do {
        dz = dz0 * TVIRT(t, h2o);
        p /= pfac;
        t = theta / pow(1000. / p, 0.286);
        intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
                            &t_env, ci, cw, 1);
        intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
                            &h2o_env, ci, cw, 0);
        dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
          TVIRT(t_env, h2o_env) * dz;
        if (dcape < 0)
          met->cin[ix][iy] += fabsf((float) dcape);
      } while (p > met->plcl[ix][iy]);

      /* Calculate level of free convection (LFC), equilibrium level (EL),
         and convective available potential energy (CAPE)... */
      dcape = 0;
      p = met->plcl[ix][iy];
      t = theta / pow(1000. / p, 0.286);
      ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
      do {
        dz = dz0 * TVIRT(t, h2o);
        p /= pfac;
        t -= lapse_rate(t, h2o) * dz;
        double psat = PSAT(t);
        h2o = psat / (p - (1. - EPS) * psat);
        intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
                            &t_env, ci, cw, 1);
        intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
                            &h2o_env, ci, cw, 0);
        double dcape_old = dcape;
        dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
          TVIRT(t_env, h2o_env) * dz;
        if (dcape > 0) {
          met->cape[ix][iy] += (float) dcape;
          if (!isfinite(met->plfc[ix][iy]))
            met->plfc[ix][iy] = (float) p;
        } else if (dcape_old > 0)
          met->pel[ix][iy] = (float) p;
        if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
          met->cin[ix][iy] += fabsf((float) dcape);
      } while (p > ptop);

      /* Check results... */
      if (!isfinite(met->plfc[ix][iy]))
        met->cin[ix][iy] = GSL_NAN;
    }
}
```

Here is the call graph for this function:



5.21.2.42 read_met_cloud() void read_met_cloud (
ctl_t * ctl,
met_t * met)

Calculate cloud properties.

Definition at line 3201 of file libtrac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
  LOG(2, "Calculate cloud data...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

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

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

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

        /* Check ice water and liquid water content... */
        if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
            || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {

          /* Get cloud top pressure ... */
          met->pct[ix][iy]
            = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));

          /* Get cloud bottom pressure ... */
          if (!isfinite(met->pcb[ix][iy]))
            met->pcb[ix][iy]
              = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
        }

        /* 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);
      }
    }
}
```

5.21.2.43 read_met_detrend() void read_met_detrend (
ctl_t * ctl,
met_t * met)

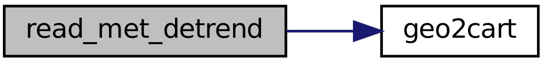
Apply detrending method to temperature and winds.

Definition at line 3251 of file libtrac.c.

```
                  {

  met_t *help;

  /* Check parameters... */
  if (ctl->met_detrend <= 0)
    return;

  /* Set timer... */
  SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
  LOG(2, "Detrend meteo data...");

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

  /* Calculate standard deviation... */
```

```
  double sigma = ctl->met_detrend / 2.355;
  double tssq = 2. * SQR(sigma);

  /* Calculate box size in latitude... */
  int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
  sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);

  /* Calculate background... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++) {
    for (int iy = 0; iy < met->ny; iy++) {

      /* Calculate Cartesian coordinates... */
      double x0[3];
      geo2cart(0.0, met->lon[ix], met->lat[iy], x0);

      /* Calculate box size in longitude... */
      int sx =
        (int) (3. * DX2DEG(sigma, met->lat[iy]) /
               fabs(met->lon[1] - met->lon[0]));
      sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);

      /* Init... */
      float wsum = 0;
      for (int ip = 0; ip < met->np; ip++) {
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
      }

      /* Loop over neighboring grid points... */
      for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
        int ix3 = ix2;
        if (ix3 < 0)
          ix3 += met->nx;
        else if (ix3 >= met->nx)
          ix3 -= met->nx;
        for (int iy2 = GSL_MAX(iy - sy, 0);
             iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {

          /* Calculate Cartesian coordinates... */
          double x1[3];
          geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);

          /* Calculate weighting factor... */
          float w = (float) exp(-DIST2(x0, x1) / tssq);

          /* Add data... */
          wsum += w;
          for (int ip = 0; ip < met->np; ip++) {
            help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
            help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
            help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
            help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
          }
        }
      }

      /* Normalize... */
      for (int ip = 0; ip < met->np; ip++) {
        help->t[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Subtract background... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++) {
        met->t[ix][iy][ip] -= help->t[ix][iy][ip];
        met->u[ix][iy][ip] -= help->u[ix][iy][ip];
        met->v[ix][iy][ip] -= help->v[ix][iy][ip];
        met->w[ix][iy][ip] -= help->w[ix][iy][ip];
      }

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

Here is the call graph for this function:



5.21.2.44 read_met_extrapolate()

```
void read_met_extrapolate (
    met_t * met )
```

Extrapolate meteo data at lower boundary.

Definition at line 3355 of file libtrac.c.

```
                    {

  /* Set timer... */
  SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
  LOG(2, "Extrapolate meteo data...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      int ip0;
      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 (int 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];
      }
    }
}
```

5.21.2.45 read_met_geopot()

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

Calculate geopotential heights.

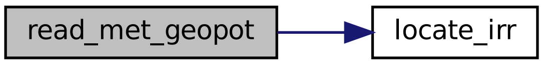
Definition at line 3392 of file libtrac.c.

```
                    {

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

```

```
  double logp[EP];

  int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;

  /* Set timer... */
  SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
  LOG(2, "Calculate geopotential heights...");

  /* Calculate log pressure... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < met->np; ip++)
    logp[ip] = log(met->p[ip]);

  /* Apply hydrostatic equation to calculate geopotential heights... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Get surface height and pressure... */
      double zs = met->zs[ix][iy];
      double lnps = log(met->ps[ix][iy]);

      /* Get temperature and water vapor vmr at the surface... */
      int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
      double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
                      met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
      double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
                        met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (zs +
                   ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
                         met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
      for (int ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] +
                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
                           met->h2o[ix][iy][ip - 1], logp[ip],
                           met->t[ix][iy][ip], met->h2o[ix][iy][ip]));

      /* Lower part of profile... */
      met->z[ix][iy][ip0]
        = (float) (zs +
                   ZDIFF(lnps, ts, h2os, logp[ip0],
                         met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
      for (int ip = ip0 - 1; ip >= 0; ip--)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip + 1] +
                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
                           met->h2o[ix][iy][ip + 1], logp[ip],
                           met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
    }

  /* Check control parameters... */
  if (dx == 0 || dy == 0)
    return;

  /* Default smoothing parameters... */
  if (dx < 0 || dy < 0) {
    if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
      dx = 3;
      dy = 2;
    } else {
      dx = 6;
      dy = 4;
    }
  }

  /* Calculate weights for smoothing... */
  float ws[dx + 1][dy + 1];
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix <= dx; ix++)
    for (int iy = 0; iy < dy; iy++)
      ws[ix][iy] = (1.0f - (float) ix / (float) dx)
        * (1.0f - (float) iy / (float) dy);

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        help[ip][ix][iy] = met->z[ix][iy][ip];

  /* Horizontal smoothing... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ip = 0; ip < met->np; ip++)
```

```
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        float res = 0, wsum = 0;
        int iy0 = GSL_MAX(iy - dy + 1, 0);
        int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
        for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
          int ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (int iy2 = iy0; iy2 <= iy1; ++iy2)
            if (isfinite(help[ip][ix3][iy2])) {
              float w = ws[abs(ix - ix2)][abs(iy - iy2)];
              res += w * help[ip][ix3][iy2];
              wsum += w;
            }
        }
        if (wsum > 0)
          met->z[ix][iy][ip] = res / wsum;
        else
          met->z[ix][iy][ip] = GSL_NAN;
      }
}
```

Here is the call graph for this function:



5.21.2.46 read_met_grid()

```
void read_met_grid (
    char * filename,
    int ncid,
    ctl_t * ctl,
    met_t * met )
```

Read coordinates of meteo data.

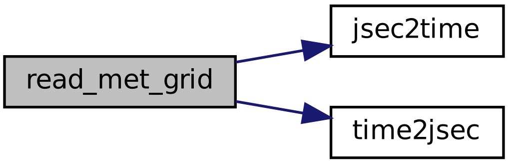
Definition at line 3512 of file libtrac.c.

```
               {

  char levname[LEN], tstr[10];

  double rtime, r2;

  int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;

  size_t np;

  /* Set timer... */
  SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
  LOG(2, "Read meteo grid information...");

  /* MPTRAC meteo files... */
  if (ctl->clams_met_data == 0) {

    /* Get time from filename... */
    size_t len = strlen(filename);
    sprintf(tstr, "%.4s", &filename[len - 16]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[len - 11]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[len - 8]);
```

```
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[len - 5]);
    hour = atoi(tstr);
    time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

    /* Check time information from data file... */
    if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
      NC(nc_get_var_double(ncid, varid, &rtime));
      if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
        WARN("Time information in meteo file does not match filename!");
    } else
      WARN("Time information in meteo file is missing!");
  }

  /* CLaMS meteo files... */
  else {

    /* Read time from file... */
    NC_GET_DOUBLE("time", &rtime, 0);

    /* Get time from filename (considering the century)... */
    if (rtime < 0)
      sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
    else
      sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
    hour = atoi(tstr);
    time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
  }

  /* Check time... */
  if (year < 1900 || year > 2100 || mon < 1 || mon > 12
      || day < 1 || day > 31 || hour < 0 || hour > 23)
    ERRMSG("Cannot read time from filename!");
  jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
  LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
      met->time, year2, mon2, day2, hour2, min2);

  /* Get grid dimensions... */
  NC_INQ_DIM("lon", &met->nx, 2, EX);
  LOG(2, "Number of longitudes: %d", met->nx);

  NC_INQ_DIM("lat", &met->ny, 2, EY);
  LOG(2, "Number of latitudes: %d", met->ny);

  if (ctl->vert_coord_met == 0) {
    int dimid;
    sprintf(levname, "lev");
    if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR)
      sprintf(levname, "plev");
  } else
    sprintf(levname, "hybrid");
  NC_INQ_DIM(levname, &met->np, 1, EP);
  if (met->np == 1) {
    int dimid;
    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));
    met->np = (int) np;
  }
  LOG(2, "Number of levels: %d", met->np);
  if (met->np < 2 || met->np > EP)
    ERRMSG("Number of levels out of range!");

  /* Read longitudes and latitudes... */
  NC_GET_DOUBLE("lon", met->lon, 1);
  LOG(2, "Longitudes: %g, %g ... %g deg",
      met->lon[0], met->lon[1], met->lon[met->nx - 1]);
  NC_GET_DOUBLE("lat", met->lat, 1);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      met->lat[0], met->lat[1], met->lat[met->ny - 1]);

  /* Read pressure levels... */
  if (ctl->met_np <= 0) {
    NC_GET_DOUBLE(levname, met->p, 1);
    for (int ip = 0; ip < met->np; ip++)
      met->p[ip] /= 100.;
    LOG(2, "Altitude levels: %g, %g ... %g km",
        Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
```

```
    LOG(2, "Pressure levels: %g, %g ... %g hPa",
        met->p[0], met->p[1], met->p[met->np - 1]);
  }
}
```

Here is the call graph for this function:



5.21.2.47 read_met_levels()

```
void read_met_levels (
    int ncid,
    ctl_t * ctl,
    met_t * met )
```

Read meteo data on vertical levels.

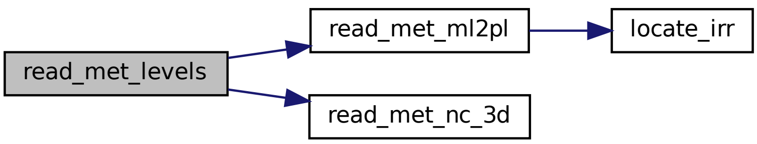
Definition at line 3634 of file libtrac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
  LOG(2, "Read level data...");

  /* MPTRAC meteo data... */
  if (ctl->clams_met_data == 0) {

    /* Read meteo data... */
    if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
      ERRMSG("Cannot read temperature!");
    if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
      ERRMSG("Cannot read zonal wind!");
    if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
      ERRMSG("Cannot read meridional wind!");
    if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
      WARN("Cannot read vertical velocity!");
    if (!read_met_nc_3d
        (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
      WARN("Cannot read specific humidity!");
    if (!read_met_nc_3d
        (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
      WARN("Cannot read ozone data!");
    if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
        WARN("Cannot read cloud liquid water content!");
      if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
        WARN("Cannot read cloud ice water content!");
    }
    if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d
          (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud rain water content!");
      if (!read_met_nc_3d
          (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud snow water content!");
    }
```

```
    if (ctl->met_relhum) {
      if (!read_met_nc_3d(ncid, "rh", "RH", ctl, met, met->h2o, 0.01f, 1))
        WARN("Cannot read relative humidity!");
#pragma omp parallel for default(shared) collapse(2)
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          for (int ip = 0; ip < met->np; ip++) {
            double pw = met->h2o[ix][iy][ip] * PSAT(met->t[ix][iy][ip]);
            met->h2o[ix][iy][ip] =
              (float) (pw / (met->p[ip] - (1.0 - EPS) * pw));
          }
    }

    /* Transfer from model levels to pressure levels... */
    if (ctl->met_np > 0) {

      /* Read pressure on model levels... */
      if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->pl, 0.01f, 1))
        ERRMSG("Cannot read pressure on model levels!");

      /* Vertical interpolation from model 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 new pressure levels... */
      met->np = ctl->met_np;
      for (int ip = 0; ip < met->np; ip++)
        met->p[ip] = ctl->met_p[ip];
    }

  }

  /* CLaMS meteo data... */
  else if (ctl->clams_met_data == 1) {

    /* Read meteorological data... */
    if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
      ERRMSG("Cannot read temperature!");
    if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
      ERRMSG("Cannot read zonal wind!");
    if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
      ERRMSG("Cannot read meridional wind!");
    if (!read_met_nc_3d(ncid, "W", "OMEGA", ctl, met, met->w, 0.01f, 1))
      WARN("Cannot read vertical velocity!");
    if (!read_met_nc_3d(ncid, "ZETA", "zeta", ctl, met, met->zeta, 1.0, 1))
      WARN("Cannot read ZETA in meteo data!");
    if (ctl->vert_vel == 1) {
      if (!read_met_nc_3d
          (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
           0.00001157407f, 1)) {
        if (!read_met_nc_3d
            (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
             0.00001157407f, 1)) {
          WARN("Cannot read vertical velocity!");
        }
      }
    }
    if (!read_met_nc_3d
        (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
      WARN("Cannot read specific humidity!");
    if (!read_met_nc_3d
        (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
      WARN("Cannot read ozone data!");
    if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
        WARN("Cannot read cloud liquid water content!");
      if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
        WARN("Cannot read cloud ice water content!");
    }
    if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d
          (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud rain water content!");
      if (!read_met_nc_3d
          (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud snow water content!");
    }

    /* Transfer from model levels to pressure levels... */
```

```
    if (ctl->met_np > 0) {

      /* Read pressure on model levels... */
      if (!read_met_nc_3d(ncid, "pl", "PRESS", ctl, met, met->pl, 1.0, 1))
        ERRMSG("Cannot read pressure on model levels!");

      /* Vertical interpolation from model 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);
      if (ctl->vert_vel == 1) {
        read_met_ml2pl(ctl, met, met->zeta);
        read_met_ml2pl(ctl, met, met->zeta_dot);
      }

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

      /* Create a pressure field... */
      for (int i = 0; i < met->nx; i++)
        for (int j = 0; j < met->ny; j++)
          for (int k = 0; k < met->np; k++) {
            met->patp[i][j][k] = (float) met->p[k];
          }
    }
  } else
    ERRMSG("Meteo data format unknown!");

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

Here is the call graph for this function:



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

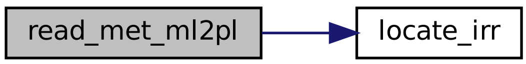
Convert meteo data from model levels to pressure levels.

Definition at line 3807 of file libtrac.c.

```
                         {

  double aux[EP], p[EP];

  /* Set timer... */
  SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
  LOG(2, "Interpolate meteo data to pressure levels...");
```

```
  /* Loop over columns... */
#pragma omp parallel for default(shared) private(aux,p) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

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

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

Here is the call graph for this function:



5.21.2.49 read_met_nc_2d()

```
int read_met_nc_2d (
    int ncid,
    char * varname,
    char * varname2,
    ctl_t * ctl,
    met_t * met,
    float dest[EX][EY],
    float scl,
    int init )
```

Read and convert 2D variable from meteo data file.

Definition at line 3848 of file libtrac.c.

```
                  {

  char varsel[LEN];

  float offset, scalfac;

  int varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
      WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
      return 0;
    } else {
```

```
      sprintf(varsel, "%s", varname2);
  } else
    sprintf(varsel, "%s", varname);

  /* Read packed data... */
  if (ctl->met_nc_scale
      && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
      && nc_get_att_float(ncid, varid, "scale_factor",
                          &scalfac) == NC_NOERR) {

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

    /* Read fill value and missing value... */
    short fillval, missval;
    if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 2-D variable: %s"
        " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
        varsel, fillval, missval, scalfac, offset);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        if (init)
          dest[ix][iy] = 0;
        short aux = help[ARRAY_2D(iy, ix, met->nx)];
        if ((fillval == 0 || aux != fillval)
            && (missval == 0 || aux != missval)
            && fabsf(aux * scalfac + offset) < 1e14f)
          dest[ix][iy] += scl * (aux * scalfac + offset);
        else
          dest[ix][iy] = GSL_NAN;
      }

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

  /* Unpacked data... */
  else {

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

    /* Read fill value and missing value... */
    float fillval, missval;
    if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
        varsel, fillval, missval);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        if (init)
          dest[ix][iy] = 0;
        float aux = help[ARRAY_2D(iy, ix, met->nx)];
        if ((fillval == 0 || aux != fillval)
            && (missval == 0 || aux != missval)
            && fabsf(aux) < 1e14f)
          dest[ix][iy] += scl * aux;
        else
          dest[ix][iy] = GSL_NAN;
      }

    /* Free... */
```

```
    free(help);
  }

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

5.21.2.50 read_met_nc_3d()

```
int read_met_nc_3d (
    int ncid,
    char * varname,
    char * varname2,
    ctl_t * ctl,
    met_t * met,
    float dest[EX][EY][EP],
    float scl,
    int init )
```

Read and convert 3D variable from meteo data file.

Definition at line 3966 of file libtrac.c.

```
            {

  char varsel[LEN];

  float offset, scalfac;

  int varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
      WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
      return 0;
    } else {
      sprintf(varsel, "%s", varname2);
  } else
    sprintf(varsel, "%s", varname);

  /* Read packed data... */
  if (ctl->met_nc_scale
      && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
      && nc_get_att_float(ncid, varid, "scale_factor",
                          &scalfac) == NC_NOERR) {

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

    /* Read fill value and missing value... */
    short fillval, missval;
    if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 3-D variable: %s "
        "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
        varsel, fillval, missval, scalfac, offset);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        for (int ip = 0; ip < met->np; ip++) {
          if (init)
            dest[ix][iy][ip] = 0;
          short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
          if ((fillval == 0 || aux != fillval)
              && (missval == 0 || aux != missval)
              && fabsf(aux * scalfac + offset) < 1e14f)
```

```
            dest[ix][iy][ip] += scl * (aux * scalfac + offset);
          else
            dest[ix][iy][ip] = GSL_NAN;
        }

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

  /* Unpacked data... */
  else {

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

    /* Read fill value and missing value... */
    float fillval, missval;
    if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
        varsel, fillval, missval);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        for (int ip = 0; ip < met->np; ip++) {
          if (init)
            dest[ix][iy][ip] = 0;
          float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
          if ((fillval == 0 || aux != fillval)
              && (missval == 0 || aux != missval)
              && fabsf(aux) < 1e14f)
            dest[ix][iy][ip] += scl * aux;
          else
            dest[ix][iy][ip] = GSL_NAN;
        }

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

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

5.21.2.51 read_met_pbl() `void read_met_pbl ( met_t * met )`

Calculate pressure of the boundary layer.

Definition at line 4086 of file libtrac.c.

```
                 {

  /* Set timer... */
  SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
  LOG(2, "Calculate planetary boundary layer...");

  /* Parameters used to estimate the height of the PBL
     (e.g., Vogelezang and Holtslag, 1996; Seidel et al., 2012)... */
  const double rib_crit = 0.25, dz = 0.05, umin = 5.0;

  /* Loop over grid points... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Set bottom level of PBL... */
      double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);

```

```
      /* Find lowest level near the bottom... */
      int ip;
      for (ip = 1; ip < met->np; ip++)
        if (met->p[ip] < pbl_bot)
          break;

      /* Get near surface data... */
      double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
                      met->p[ip], met->z[ix][iy][ip], pbl_bot);
      double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
                      met->p[ip], met->t[ix][iy][ip], pbl_bot);
      double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
                      met->p[ip], met->u[ix][iy][ip], pbl_bot);
      double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
                      met->p[ip], met->v[ix][iy][ip], pbl_bot);
      double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
                        met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
      double tvs = THETAVIRT(pbl_bot, ts, h2os);

      /* Init... */
      double rib_old = 0;

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

        /* Get squared horizontal wind speed... */
        double vh2
          = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
        vh2 = GSL_MAX(vh2, SQR(umin));

        /* Calculate bulk Richardson number... */
        double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
          * (THETAVIRT(met->p[ip], met->t[ix][iy][ip],
                       met->h2o[ix][iy][ip]) - tvs) / vh2;

        /* Check for critical value... */
        if (rib >= rib_crit) {
          met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
                                          rib, met->p[ip], rib_crit));
          if (met->pbl[ix][iy] > pbl_bot)
            met->pbl[ix][iy] = (float) pbl_bot;
          break;
        }

        /* Save Richardson number... */
        rib_old = rib;
      }
    }
}
```

5.21.2.52 read_met_periodic()

```
void read_met_periodic (
    met_t * met )
```

Create meteo data with periodic boundary conditions.

Definition at line 4157 of file libtrac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
  LOG(2, "Apply periodic boundary conditions...");

  /* 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];
```

```
    met->ts[met->nx - 1][iy] = met->ts[0][iy];
    met->us[met->nx - 1][iy] = met->us[0][iy];
    met->vs[met->nx - 1][iy] = met->vs[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];
    }
  }
}
```

5.21.2.53 read_met_pv() void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 4199 of file libtrac.c.

```
                {

  double pows[EP];

  /* Set timer... */
  SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
  LOG(2, "Calculate potential vorticity...");

  /* Set powers... */
#pragma omp parallel for default(shared)
  for (int 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)
  for (int ix = 0; ix < met->nx; ix++) {

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

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

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

      /* Set auxiliary variables... */
      double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
      double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
      double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
      double c0 = cos(met->lat[iy0] / 180. * M_PI);
      double c1 = cos(met->lat[iy1] / 180. * M_PI);
      double cr = cos(latr / 180. * M_PI);
      double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);

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

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

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

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

        /* Get gradients in pressure... */
        double dtdp, dudp, dvdp;
```

```
        double dp0 = 100. * (met->p[ip] - met->p[ip0]);
        double dp1 = 100. * (met->p[ip1] - met->p[ip]);
        if (ip != ip0 && ip != ip1) {
          double 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 {
          double 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)
  for (int ix = 0; ix < met->nx; ix++)
    for (int 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];
    }
}
```

5.21.2.54 read_met_sample()

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

Downsampling of meteo data.

Definition at line 4305 of file libtrac.c.

```
                  {

  met_t *help;

  /* 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;

  /* Set timer... */
  SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
  LOG(2, "Downsampling of meteo data...");

  /* 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 (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
    for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
      for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
        help->ps[ix][iy] = 0;
        help->zs[ix][iy] = 0;
        help->ts[ix][iy] = 0;
        help->us[ix][iy] = 0;
        help->vs[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;
        float wsum = 0;
        for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
             ix2++) {
          int ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
            for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
              float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
                * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
                * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->zs[ix][iy] += w * met->zs[ix3][iy2];
              help->ts[ix][iy] += w * met->ts[ix3][iy2];
              help->us[ix][iy] += w * met->us[ix3][iy2];
              help->vs[ix][iy] += w * met->vs[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->ts[ix][iy] /= wsum;
        help->us[ix][iy] /= wsum;
        help->vs[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 (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (int 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->ts[met->nx][met->ny] = help->ts[ix][iy];
      met->us[met->nx][met->ny] = help->us[ix][iy];
      met->vs[met->nx][met->ny] = help->vs[ix][iy];
      met->np = 0;
      for (int 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);
}
```

5.21.2.55 read_met_surface()

```
void read_met_surface (
    int ncid,
    met_t * met,
    ctl_t * ctl )
```

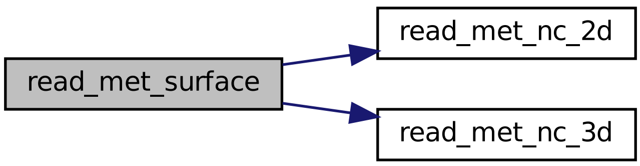
Read surface data.

Definition at line 4433 of file libtrac.c.

```
                      {

  /* Set timer... */
  SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
  LOG(2, "Read surface data...");

  /* MPTRAC meteo data... */
  if (ctl->clams_met_data == 0) {

    /* Read surface pressure... */
    if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
      if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
        if (!read_met_nc_2d(ncid, "sp", "SP", ctl, met, met->ps, 0.01f, 1)) {
          WARN("Cannot not read surface pressure data (use lowest level)!");
          for (int ix = 0; ix < met->nx; ix++)
            for (int iy = 0; iy < met->ny; iy++)
              met->ps[ix][iy] = (float) met->p[0];
        }
      }
    } else
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);

    /* Read geopotential height at the surface... */
    if (!read_met_nc_2d
        (ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
      if (!read_met_nc_2d
          (ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
        WARN("Cannot read surface geopotential height!");

    /* Read temperature at the surface... */
    if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
      WARN("Cannot read surface temperature!");

    /* Read zonal wind at the surface... */
    if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
      WARN("Cannot read surface zonal wind!");

    /* Read meridional wind at the surface... */
    if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
      WARN("Cannot read surface meridional wind!");
  }

  /* CLaMS meteo data... */
  else {

    /* Read surface pressure... */
    if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
      WARN("Cannot not read surface pressure data (use lowest level)!");
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) met->p[0];
    }

    /* Read geopotential height at the surface
       (use lowermost level of 3-D data field)... */
    float *help;
```

```
    ALLOC(help, float,
          EX * EY * EP);
    memcpy(help, met->pl, sizeof(met->pl));
    if (!read_met_nc_3d
        (ncid, "gph", "GPH", ctl, met, met->pl, (float) (1e-3 / G0), 1)) {
      ERRMSG("Cannot read geopotential height!");
    } else
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->zs[ix][iy] = met->pl[ix][iy][0];
    memcpy(met->pl, help, sizeof(met->pl));
    free(help);

    /* Read temperature at the surface... */
    if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
      WARN("Cannot read surface temperature!");

    /* Read zonal wind at the surface... */
    if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
      WARN("Cannot read surface zonal wind!");

    /* Read meridional wind at the surface... */
    if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
      WARN("Cannot read surface meridional wind!");
  }
}
```

Here is the call graph for this function:



5.21.2.56 read_met_tropo()

```
void read_met_tropo (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met )
```

Calculate tropopause data.

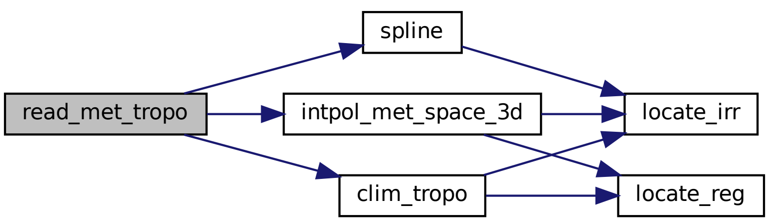
Definition at line 4523 of file libtrac.c.

```
               {

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

  /* Set timer... */
  SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
  LOG(2, "Calculate tropopause...");

  /* Get altitude and pressure profiles... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < met->np; iz++)
    z[iz] = Z(met->p[iz]);
#pragma omp parallel for default(shared)
  for (int 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)
#pragma omp parallel for default(shared) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int 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) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = (float) clim_tropo(clim, 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(t,t2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        for (int iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);

        /* Find minimum... */
        int 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(t,t2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        int iz;
        for (iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);

        /* Find 1st tropopause... */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++) {
          int found = 1;
          for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
            if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
                ctl->met_tropo_lapse) {
              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++) {
            int found = 1;
            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
                  ctl->met_tropo_lapse_sep) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 170; iz++) {
            int found = 1;
            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
```

```
                  ctl->met_tropo_lapse) {
                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(pv,pv2,th,th2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate potential vorticity profile... */
        for (int iz = 0; iz < met->np; iz++)
          pv[iz] = met->pv[ix][iy][iz];
        spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);

        /* Interpolate potential temperature profile... */
        for (int 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, ctl->met_tropo_spline);

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

  else
    ERRMSG("Cannot calculate tropopause!");

  /* Interpolate temperature, geopotential height, and water vapor vmr... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {
      double h2ot, tt, zt;
      INTPOL_INIT;
      intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &tt, ci, cw, 1);
      intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &zt, ci, cw, 0);
      intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &h2ot, ci, cw, 0);
      met->tt[ix][iy] = (float) tt;
      met->zt[ix][iy] = (float) zt;
      met->h2ot[ix][iy] = (float) h2ot;
    }
}
```

Here is the call graph for this function:



5.21.2.57 read_obs()

```
void read_obs (
    char * filename,
    double * rt,
    double * rz,
    double * rlon,
    double * rlat,
    double * robs,
    int * nobs )
```

Read observation data.

Definition at line 4698 of file libtrac.c.

```
                {

  FILE *in;

  char line[LEN];

  /* Open observation data file... */
  LOG(1, "Read observation data: %s", filename);
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read observations... */
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
               &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
      if ((++(*nobs)) >= NOBS)
        ERRMSG("Too many observations!");

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

  /* Check time... */
  for (int i = 1; i < *nobs; i++)
    if (rt[i] < rt[i - 1])
      ERRMSG("Time must be ascending!");

  /* Write info... */
  int n = *nobs;
  double mini, maxi;
  LOG(2, "Number of observations: %d", *nobs);
  gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
  LOG(2, "Observation range: %g ... %g", mini, maxi);
}
```

5.21.2.58 scan_ctl()

```
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 4749 of file libtrac.c.

```
                {

  FILE *in = NULL;

  char fullname1[LEN], fullname2[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) {
    char dummy[LEN], line[LEN], rvarname[LEN];
    while (fgets(line, LEN, in)) {
      if (sscanf(line, "%4999s %4999s %4999s", 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
      ERRMSG("Missing variable %s!\n", fullname1);
  }

  /* Write info... */
  LOG(1, "%s = %s", fullname1, rval);

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

5.21.2.59 sedi()

```
double sedi (
    double p,
    double T,
    double rp,
    double rhop )
```

Calculate sedimentation velocity.

Definition at line 4821 of file libtrac.c.

```
                {

  /* Convert particle radius from microns to m... */
  rp *= 1e-6;

```

```
  /* Density of dry air [kg / m^3]... */
  double rho = RHO(p, T);

  /* Dynamic viscosity of air [kg / (m s)]... */
  double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Thermal velocity of an air molecule [m / s]... */
  double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));

  /* Mean free path of an air molecule [m]... */
  double lambda = 2. * eta / (rho * v);

  /* Knudsen number for air (dimensionless)... */
  double K = lambda / rp;

  /* Cunningham slip-flow correction (dimensionless)... */
  double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));

  /* Sedimentation velocity [m / s]... */
  return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
}
```

5.21.2.60 spline()

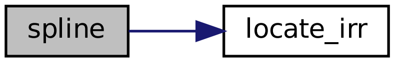
```
void spline (
    double * x,
    double * y,
    int n,
    double * x2,
    double * y2,
    int n2,
    int method )
```

Spline interpolation.

Definition at line 4854 of file libtrac.c.

```
                   {

  /* Cubic spline interpolation... */
  if (method == 1) {

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

    /* Interpolate 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);
  }

  /* Linear interpolation... */
  else {
    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 {
        int idx = locate_irr(x, n, x2[i]);
        y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
      }
  }
}
```

Here is the call graph for this function:



5.21.2.61 stddev() `float stddev ( float * data, int n )`

Calculate standard deviation.

Definition at line 4903 of file libtrac.c.

```
              {

  if (n <= 0)
    return 0;

  float mean = 0, var = 0;

  for (int i = 0; i < n; ++i) {
    mean += data[i];
    var += SQR(data[i]);
  }

  var = var / (float) n - SQR(mean / (float) n);

  return (var > 0 ? sqrtf(var) : 0);
}
```

5.21.2.62 sza() `double sza ( double sec, double lon, double lat )`

Calculate solar zenith angle.

Definition at line 4924 of file libtrac.c.

```
                   {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

```

```
  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [rad]... */
  return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
}
```

5.21.2.63 time2jsec()

```
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 4966 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;
}
```

5.21.2.64 timer()

```
void timer (
    const char * name,
    const char * group,
    int output )
```

Measure wall-clock time.

Definition at line 4997 of file libtrac.c.

```
              {

  static char names[NTIMER][100], groups[NTIMER][100];

  static double rt_name[NTIMER], rt_group[NTIMER],
```

```
    rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;

  static int iname = -1, igroup = -1, nname, ngroup, ct_name[NTIMER];

  /* Get time... */
  t1 = omp_get_wtime();
  dt = t1 - t0;

  /* Add elapsed time to current timers... */
  if (iname >= 0) {
    rt_name[iname] += dt;
    rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
    rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
    ct_name[iname]++;
  }
  if (igroup >= 0)
    rt_group[igroup] += t1 - t0;

  /* Report timers... */
  if (output) {
    for (int i = 0; i < nname; i++)
      LOG(1, "TIMER_%s = %.3f s    (min= %g s, mean= %g s,"
          " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
          rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
    for (int i = 0; i < ngroup; i++)
      LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
    double total = 0.0;
    for (int i = 0; i < nname; i++)
      total += rt_name[i];
    LOG(1, "TIMER_TOTAL = %.3f s", total);
  }

  /* Identify IDs of next timer... */
  for (iname = 0; iname < nname; iname++)
    if (strcasecmp(name, names[iname]) == 0)
      break;
  for (igroup = 0; igroup < ngroup; igroup++)
    if (strcasecmp(group, groups[igroup]) == 0)
      break;

  /* Check whether this is a new timer... */
  if (iname >= nname) {
    sprintf(names[iname], "%s", name);
    if ((++nname) > NTIMER)
      ERRMSG("Too many timers!");
  }

  /* Check whether this is a new group... */
  if (igroup >= ngroup) {
    sprintf(groups[igroup], "%s", group);
    if ((++ngroup) > NTIMER)
      ERRMSG("Too many groups!");
  }

  /* Save starting time... */
  t0 = t1;
}
```

5.21.2.65 tropo_weight()

```
double tropo_weight (
        clim_t * clim,
        double t,
        double lat,
        double p )
```

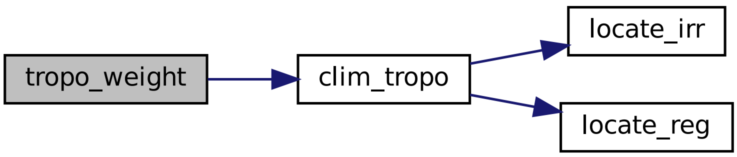
Get weighting factor based on tropopause distance.

Definition at line 5065 of file libtrac.c.

```
                 {

  /* Get tropopause pressure... */
  double pt = clim_tropo(clim, t, lat);

  /* Get pressure range... */
  double p1 = pt * 0.866877899;
  double p0 = pt / 0.866877899;

  /* Get weighting factor... */
```

```
  if (p > p0)
    return 1;
  else if (p < p1)
    return 0;
  else
    return LIN(p0, 1.0, p1, 0.0, p);
}
```

Here is the call graph for this function:



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

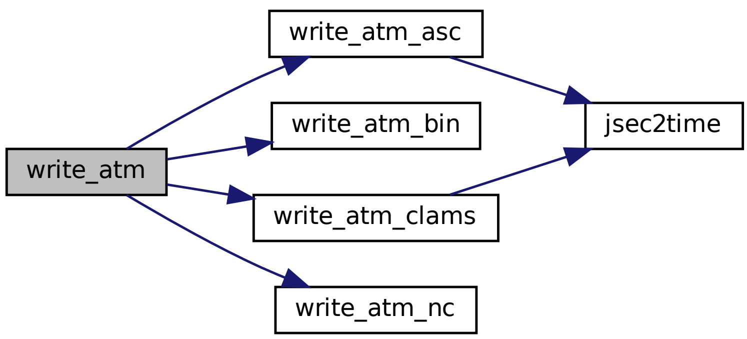
Write atmospheric data.

Definition at line 5089 of file libtrac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write atmospheric data: %s", filename);

  /* Write ASCII data... */
  if (ctl->atm_type == 0)
    write_atm_asc(filename, ctl, atm, t);

  /* Write binary data... */
  else if (ctl->atm_type == 1)
    write_atm_bin(filename, ctl, atm);

  /* Write netCDF data... */
  else if (ctl->atm_type == 2)
    write_atm_nc(filename, ctl, atm);

  /* Write CLaMS data... */
  else if (ctl->atm_type == 3)
    write_atm_clams(ctl, atm, t);

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of particles: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
```

```
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  for (int iq = 0; iq < ctl->nq; iq++) {
    char msg[LEN];
    sprintf(msg, "Quantity %s range: %s ... %s %s",
            ctl->qnt_name[iq], ctl->qnt_format[iq],
            ctl->qnt_format[iq], ctl->qnt_unit[iq]);
    gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
    LOG(2, msg, mini, maxi);
  }
}
```

Here is the call graph for this function:



5.21.2.67 write_atm_asc() void write_atm_asc (

```
        const char * filename,
        ctl_t * ctl,
        atm_t * atm,
        double t )
```

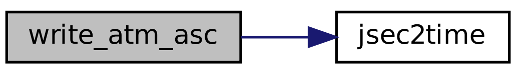
Write atmospheric data in ASCII format.

Definition at line 5145 of file libtrac.c.

```
                {

  FILE *out;

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

  /* 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... */
    double r;
    int year, mon, day, hour, min, sec;
    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... */
    FILE *in;
    if (!(in = fopen(ctl->atm_gpfile, "r")))
      ERRMSG("Cannot open file!");
    char line[LEN];
    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 (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");

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

    /* Check time... */
    if (ctl->atm_filter == 2 && (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 (int iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
        fprintf(out, ctl->qnt_format[iq], GSL_NAN);
      else
        fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

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

Here is the call graph for this function:



5.21.2.68 write_atm_bin()

```
void write_atm_bin (
    const char * filename,
    ctl_t * ctl,
    atm_t * atm )
```

Write atmospheric data in binary format.

Definition at line 5227 of file libtrac.c.

```
                    {

  FILE *out;

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

  /* Write version of binary data... */
  int version = 100;
  FWRITE(&version, int,
         1,
         out);

  /* 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 (int iq = 0; iq < ctl->nq; iq++)
    FWRITE(atm->q[iq], double,
             (size_t) atm->np,
           out);

  /* Write final flag... */
  int final = 999;
  FWRITE(&final, int,
         1,
         out);

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

5.21.2.69 write_atm_clams()

```
void write_atm_clams (
    ctl_t * ctl,
    atm_t * atm,
    double t )
```

Write atmospheric data in CLaMS format.

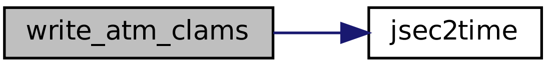
Definition at line 5277 of file libtrac.c.

```
                    {

  /* Global Counter... */
  static size_t out_cnt = 0;

  char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";

  double r, r_start, r_stop;

  int year, mon, day, hour, min, sec;
  int year_start, mon_start, day_start, hour_start, min_start, sec_start;
  int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
  int ncid, varid, tid, pid, cid, zid, dim_ids[2];

  /* time, nparc */
  size_t start[2], count[2];

  /* Determine start and stop times of calculation... */
  jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
  jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
            &min_start, &sec_start, &r_start);
  jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
            &min_stop, &sec_stop, &r_stop);

```

```
/* Set filename... */
sprintf(filename_out,
        "./traj_fix_3d_%02d%02d%02d%02d_%02d%02d%02d%02d.nc",
        year_start % 100, mon_start, day_start, hour_start,
        year_stop % 100, mon_stop, day_stop, hour_stop);
printf("Write traj file: %s\n", filename_out);

/* Define hyperslap for the traj_file... */
start[0] = out_cnt;
start[1] = 0;
count[0] = 1;
count[1] = (size_t) atm->np;

/* Create the file at the first timestep... */
if (out_cnt == 0) {

  /* Create file... */
  nc_create(filename_out, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
  NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
  NC(nc_def_dim(ncid, "TMDT", 7, &cid));
  dim_ids[0] = tid;
  dim_ids[1] = pid;

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
  NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
  NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
  NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
               ctl->qnt_name[iq], ctl->qnt_unit[iq]);

  /* Define global attributes... */
  NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
  NC_PUT_ATT_GLOBAL("model", "MPTRAC");

  /* End definitions... */
  NC(nc_enddef(ncid));
  NC(nc_close(ncid));
}

/* Increment global counter to change hyperslap... */
out_cnt++;

/* Open file... */
NC(nc_open(filename_out, NC_WRITE, &ncid));

/* Write data... */
NC_PUT_DOUBLE("time", atm->time, 1);
NC_PUT_DOUBLE("LAT", atm->lat, 1);
NC_PUT_DOUBLE("LON", atm->lon, 1);
NC_PUT_DOUBLE("PRESS", atm->p, 1);
if (ctl->vert_coord_ap == 1) {
  NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
} else if (ctl->qnt_zeta >= 0) {
  NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
}
for (int iq = 0; iq < ctl->nq; iq++)
  NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);

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

/* At the last time step create the init_fix_YYYYMMDDHH file... */
if ((year == year_stop) && (mon == mon_stop)
    && (day == day_stop) && (hour == hour_stop)) {

  /* Set filename... */
  char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
  sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
          year_stop % 100, mon_stop, day_stop, hour_stop);
  printf("Write init file: %s\n", filename_init);

  /* Create file... */
  nc_create(filename_init, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "time", 1, &tid));
  NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
  dim_ids[0] = tid;
  dim_ids[1] = pid;

```

```
    /* Define variables and their attributes... */
    NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
               "seconds since 2000-01-01 00:00:00 UTC");
    NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
    NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
    NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
    NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
    NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
    NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);

    /* Define global attributes... */
    NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
    NC_PUT_ATT_GLOBAL("model", "MPTRAC");

    /* End definitions... */
    NC(nc_enddef(ncid));

    /* Write data... */
    NC_PUT_DOUBLE("time", atm->time, 0);
    NC_PUT_DOUBLE("LAT", atm->lat, 0);
    NC_PUT_DOUBLE("LON", atm->lon, 0);
    NC_PUT_DOUBLE("PRESS", atm->p, 0);
    NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

Here is the call graph for this function:



5.21.2.70 write_atm_nc()

```
void write_atm_nc (
    const char * filename,
    ctl_t * ctl,
    atm_t * atm )
```

Write atmospheric data in netCDF format.

Definition at line 5427 of file libtrac.c.

```
               {

  int ncid, obsid, varid;

  size_t start[2], count[2];

  /* Create file... */
  nc_create(filename, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
```

```c
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
               ctl->qnt_longname[iq], ctl->qnt_unit[iq]);

  /* Define global attributes... */
  NC_PUT_ATT_GLOBAL("featureType", "point");

  /* End definitions... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", atm->time, 0);
  NC_PUT_DOUBLE("press", atm->p, 0);
  NC_PUT_DOUBLE("lon", atm->lon, 0);
  NC_PUT_DOUBLE("lat", atm->lat, 0);
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

5.21.2.71 write_csi()

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

Write CSI data.

Definition at line 5472 of file libtrac.c.

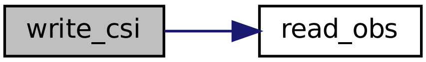
```c
                {

  static FILE *out;

  static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
    dlon, dlat, dz, x[NCSI], y[NCSI];

  static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;

  /* Set timer... */
  SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);

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

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

    /* Allocate... */
    ALLOC(area, double,
          ctl->csi_ny);
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create new file... */
    LOG(1, "Write CSI data: %s", 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 (ratio of forecasts and observations) [%%]\n"
            "# $8 = probability of detection (POD) [%%]\n"
            "# $9 = false alarm rate (FAR) [%%]\n"
            "# $10 = critical success index (CSI) [%%]\n");
    fprintf(out,
            "# $11 = hits associated with random chance\n"
            "# $12 = equitable threat score (ETS) [%%]\n"
            "# $13 = Pearson linear correlation coefficient\n"
            "# $14 = Spearman rank-order correlation coefficient\n"
            "# $15 = column density mean error (F - O) [kg/m^2]\n"
            "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
            "# $17 = column density mean absolute error [kg/m^2]\n"
            "# $18 = number of data points\n\n");

    /* Set grid box size... */
    dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
    dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
    dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;

    /* Set horizontal coordinates... */
    for (iy = 0; iy < ctl->csi_ny; iy++) {
      double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
      area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
    }
  }

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

  /* Allocate... */
  ALLOC(modmean, double,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
  ALLOC(obsmean, double,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
  ALLOC(obscount, int,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Check observation data... */
    if (!isfinite(robs[i]))
      continue;

    /* Calculate indices... */
    ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
    iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
    iz = (int) ((rz[i] - ctl->csi_z0) / dz);

    /* 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... */
    int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
    obsmean[idx] += robs[i];
    obscount[idx]++;
  }

  /* Analyze model data... */
  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) / dlon);
    iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
    iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);

    /* 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... */
    int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
    modmean[idx] += atm->q[ctl->qnt_m][ip];
  }

  /* Analyze all grid cells... */
  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... */
        int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
        if (obscount[idx] > 0)
          obsmean[idx] /= obscount[idx];

        /* Calculate column density... */
        if (modmean[idx] > 0)
          modmean[idx] /= (1e6 * area[iy]);

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

        /* Save data for other verification statistics... */
        if (obscount[idx] > 0
            && (obsmean[idx] >= ctl->csi_obsmin
                || modmean[idx] >= ctl->csi_modmin)) {
          x[n] = modmean[idx];
          y[n] = obsmean[idx];
          if ((++n) > NCSI)
            ERRMSG("Too many data points to calculate statistics!");
        }
      }

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

    /* Calculate verification statistics
       (https://www.cawcr.gov.au/projects/verification/) ... */
    static double work[2 * NCSI];
    int n_obs = cx + cy;
    int n_for = cx + cz;
    double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
    double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
    double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
    double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
    double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
    double ets = (cx + cy + cz - cx_rd > 0) ?
      (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
    double rho_p =
      (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
    double rho_s =
      (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
    for (int i = 0; i < n; i++)
      work[i] = x[i] - y[i];
    double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
    double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
                                                          0.0) : GSL_NAN;
    double absdev =
      (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;

    /* Write... */
    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %g %g %d\n",
            t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
            rho_p, rho_s, mean, rmse, absdev, n);

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

  /* Free... */
  free(modmean);
  free(obsmean);
  free(obscount);

```

```
  /* Finalize... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(area);
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}
```

Here is the call graph for this function:



5.21.2.72 write_ens()

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

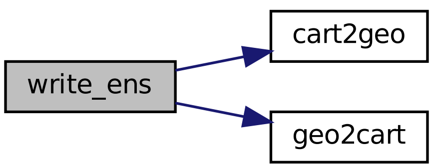
Write ensemble data.

Definition at line 5712 of file libtrac.c.

```
                {

  static FILE *out;

  static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
    x[3], zm[NENS];

  static int n[NENS];

  /* Set timer... */
  SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);

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

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

  /* Init... */
  for (int i = 0; i < NENS; i++) {
    for (int iq = 0; iq < ctl->nq; iq++)
      qm[iq][i] = qs[iq][i] = 0;
    xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
    n[i] = 0;
  }

  /* 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 ensemble ID... */
    if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
      ERRMSG("Ensemble ID is out of range!");

    /* Get means... */
    geo2cart(0, atm->lon[ip], atm->lat[ip], x);
    for (int iq = 0; iq < ctl->nq; iq++) {
      qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
      qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
    }
    xm[ctl->qnt_ens][0] += x[0];
    xm[ctl->qnt_ens][1] += x[1];
    xm[ctl->qnt_ens][2] += x[2];
    zm[ctl->qnt_ens] += Z(atm->p[ip]);
    n[ctl->qnt_ens]++;
  }

  /* Create file... */
  LOG(1, "Write ensemble data: %s", 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 (int iq = 0; iq < ctl->nq; iq++)
    fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
            ctl->qnt_name[iq], ctl->qnt_unit[iq]);
  for (int 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);

  /* Write data... */
  for (int i = 0; i < NENS; i++)
    if (n[i] > 0) {
      cart2geo(xm[i], &dummy, &lon, &lat);
      fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
      for (int iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
      }
      for (int iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
        fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
      }
      fprintf(out, " %d\n", n[i]);
    }

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

Here is the call graph for this function:



5.21.2.73 write_grid()

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

Write gridded data.

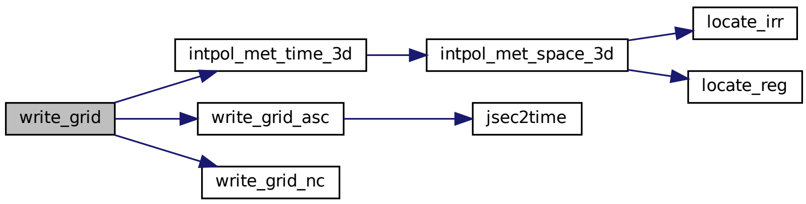
Definition at line 5809 of file libtrac.c.

```
                {

  double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;

  int *ixs, *iys, *izs, *np;

  /* Set timer... */
  SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write grid data: %s", filename);

  /* Allocate... */
  ALLOC(cd, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(mass, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(vmr_expl, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(vmr_impl, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(z, double,
        ctl->grid_nz);
  ALLOC(lon, double,
        ctl->grid_nx);
  ALLOC(lat, double,
        ctl->grid_ny);
  ALLOC(area, double,
        ctl->grid_ny);
  ALLOC(press, double,
        ctl->grid_nz);
  ALLOC(np, int,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(ixs, int,
        atm->np);
  ALLOC(iys, int,
        atm->np);
  ALLOC(izs, int,
        atm->np);

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

  /* Set vertical coordinates... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < ctl->grid_nz; iz++) {
    z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
    press[iz] = P(z[iz]);
  }

  /* Set horizontal coordinates... */
  for (int ix = 0; ix < ctl->grid_nx; ix++)
    lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
#pragma omp parallel for default(shared)
  for (int iy = 0; iy < ctl->grid_ny; iy++) {
    lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
    area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
      * cos(lat[iy] * M_PI / 180.);
  }

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

  /* Get grid box indices... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++) {
    ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
    iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
```

```
    izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
    if (atm->time[ip] < t0 || atm->time[ip] > t1
        || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
        || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
        || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
      izs[ip] = -1;
  }

  /* Average data... */
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0) {
      int idx =
        ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
      np[idx]++;
      if (ctl->qnt_m >= 0)
        mass[idx] += atm->q[ctl->qnt_m][ip];
      if (ctl->qnt_vmr >= 0)
        vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
    }

  /* Calculate column density and vmr... */
#pragma omp parallel for default(shared)
  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++) {

        /* Get grid index... */
        int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);

        /* Calculate column density... */
        cd[idx] = GSL_NAN;
        if (ctl->qnt_m >= 0)
          cd[idx] = mass[idx] / (1e6 * area[iy]);

        /* Calculate volume mixing ratio (implicit)... */
        vmr_impl[idx] = GSL_NAN;
        if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
          vmr_impl[idx] = 0;
          if (mass[idx] > 0) {

            /* Get temperature... */
            double temp;
            INTPOL_INIT;
            intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
                               lon[ix], lat[iy], &temp, ci, cw, 1);

            /* Calculate volume mixing ratio... */
            vmr_impl[idx] = MA / ctl->molmass * mass[idx]
              / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
          }
        }

        /* Calculate volume mixing ratio (explicit)... */
        if (ctl->qnt_vmr >= 0 && np[idx] > 0)
          vmr_expl[idx] /= np[idx];
        else
          vmr_expl[idx] = GSL_NAN;
      }

  /* Write ASCII data... */
  if (ctl->grid_type == 0)
    write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
                   t, z, lon, lat, area, dz, np);

  /* Write netCDF data... */
  else if (ctl->grid_type == 1)
    write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
                  t, z, lon, lat, area, dz, np);

  /* Error message... */
  else
    ERRMSG("Grid data format GRID_TYPE unknown!");

  /* Free... */
  free(cd);
  free(mass);
  free(vmr_expl);
  free(vmr_impl);
  free(z);
  free(lon);
  free(lat);
  free(area);
  free(press);
  free(np);
  free(ixs);
  free(iys);
  free(izs);
```

```
}
```

Here is the call graph for this function:



5.21.2.74 write_grid_asc()

```
void write_grid_asc (
    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )
```

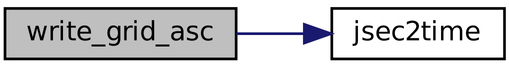
Write gridded data in ASCII format.

Definition at line 5977 of file libtrac.c.

```
            {

  FILE *in, *out;

  char line[LEN];

  /* Check if gnuplot output is requested... */
  if (ctl->grid_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... */
    double r;
    int year, mon, day, hour, min, sec;
    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 {
```

```

    /* 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 depth [km]\n"
          "# $7 = number of particles [1]\n"
          "# $8 = column density (implicit) [kg/m^2]\n"
          "# $9 = volume mixing ratio (implicit) [ppv]\n"
          "# $10 = volume mixing ratio (explicit) [ppv]\n\n");

  /* Write data... */
  for (int ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
      fprintf(out, "\n");
    for (int iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
        fprintf(out, "\n");
      for (int iz = 0; iz < ctl->grid_nz; iz++) {
        int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
        if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
          fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
                  t, z[iz], lon[ix], lat[iy], area[iy], dz,
                  np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
      }
    }
  }

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

Here is the call graph for this function:



5.21.2.75 write_grid_nc()

```
void write_grid_nc (
    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )
```

Write gridded data in netCDF format.

Definition at line 6063 of file libtrac.c.

```
        {

  double *help;

  int *help2, ncid, dimid[10], varid;

  size_t start[2], count[2];

  /* Allocate... */
  ALLOC(help, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(help2, int,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);

  /* Create file... */
  nc_create(filename, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
  NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[1]));
  NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
  NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[3]));
  NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[1], "altitude", "km");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[3], "longitude", "degrees_east");
  NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[1], "layer depth", "km");
  NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
  NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
  NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
             "volume mixing ratio (implicit)", "ppv");
  NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
             "volume mixing ratio (explicit)", "ppv");
  NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");

  /* End definitions... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", &t, 0);
  NC_PUT_DOUBLE("lon", lon, 0);
  NC_PUT_DOUBLE("lat", lat, 0);
  NC_PUT_DOUBLE("z", z, 0);
  NC_PUT_DOUBLE("area", area, 0);
  NC_PUT_DOUBLE("dz", &dz, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          cd[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("cd", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          vmr_impl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("vmr_impl", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          vmr_expl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("vmr_expl", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help2[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          np[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_INT("np", help2, 0);

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

  /* Free... */
  free(help);
  free(help2);
}
```

5.21.2.76 write_met()

```
int write_met (
    char * filename,
    ctl_t * ctl,
    met_t * met )
```

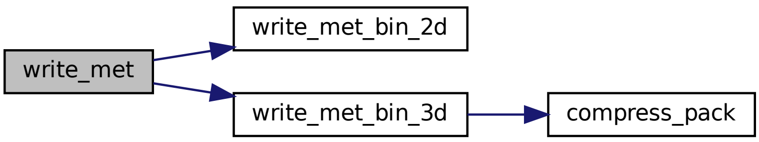
Read meteo data file.

Definition at line 6163 of file libtrac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write meteo data: %s", filename);

  /* Check compression flags... */
#ifndef ZFP
  if (ctl->met_type == 3)
    ERRMSG("zfp compression not supported!");
#endif
#ifndef ZSTD
  if (ctl->met_type == 4)
    ERRMSG("zstd compression not supported!");
#endif

  /* Write binary... */
  if (ctl->met_type >= 1 && ctl->met_type <= 4) {

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

    /* Write type of binary data... */
    FWRITE(&ctl->met_type, int,
           1,
           out);

    /* Write version of binary data... */
    int version = 100;
    FWRITE(&version, int,
           1,
           out);

    /* Write grid data... */
    FWRITE(&met->time, double,
           1,
           out);
    FWRITE(&met->nx, int,
           1,
           out);
    FWRITE(&met->ny, int,
           1,
           out);
    FWRITE(&met->np, int,
           1,
           out);
    FWRITE(met->lon, double,
             (size_t) met->nx,
           out);
    FWRITE(met->lat, double,
             (size_t) met->ny,
           out);
    FWRITE(met->p, double,
             (size_t) met->np,
           out);

    /* Write surface data... */
    write_met_bin_2d(out, met, met->ps, "PS");
    write_met_bin_2d(out, met, met->ts, "TS");
    write_met_bin_2d(out, met, met->zs, "ZS");
    write_met_bin_2d(out, met, met->us, "US");
    write_met_bin_2d(out, met, met->vs, "VS");
    write_met_bin_2d(out, met, met->pbl, "PBL");
    write_met_bin_2d(out, met, met->pt, "PT");
    write_met_bin_2d(out, met, met->tt, "TT");
    write_met_bin_2d(out, met, met->zt, "ZT");
    write_met_bin_2d(out, met, met->h2ot, "H2OT");
```

```
    write_met_bin_2d(out, met, met->pct, "PCT");
    write_met_bin_2d(out, met, met->pcb, "PCB");
    write_met_bin_2d(out, met, met->cl, "CL");
    write_met_bin_2d(out, met, met->plcl, "PLCL");
    write_met_bin_2d(out, met, met->plfc, "PLFC");
    write_met_bin_2d(out, met, met->pel, "PEL");
    write_met_bin_2d(out, met, met->cape, "CAPE");
    write_met_bin_2d(out, met, met->cin, "CIN");

    /* Write level data... */
    write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
    write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
    write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
    write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
    write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
    write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
    write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
    write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
    write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
    write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);

    /* Write final flag... */
    int final = 999;
    FWRITE(&final, int,
           1,
           out);

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

  return 0;
}
```

Here is the call graph for this function:



5.21.2.77 write_met_bin_2d() `void write_met_bin_2d ( FILE * out, met_t * met, float var[EX][EY], char * varname )`

Write 2-D meteo variable.

Definition at line 6273 of file libtrac.c.

```
                 {

  float *help;

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

  /* Copy data... */
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];
```

```

  /* Write uncompressed data... */
  LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
  FWRITE(help, float,
         (size_t) (met->nx * met->ny),
         out);

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

5.21.2.78 write_met_bin_3d()

void write_met_bin_3d (
FILE * *out*,
ctl_t * *ctl*,
met_t * *met*,
float *var[EX][EY][EP]*,
char * *varname*,
int *precision*,
double *tolerance*)

Write 3-D meteo variable.

Definition at line 6302 of file libtrac.c.

```
                          {

  float *help;

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

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];

  /* Write uncompressed data... */
  if (ctl->met_type == 1) {
    LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
    FWRITE(help, float,
           (size_t) (met->nx * met->ny * met->np),
           out);
  }

  /* Write packed data... */
  else if (ctl->met_type == 2)
    compress_pack(varname, help, (size_t) (met->ny * met->nx),
                  (size_t) met->np, 0, out);

  /* Write zfp data... */
#ifdef ZFP
  else if (ctl->met_type == 3)
    compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
                 tolerance, 0, out);
#endif

  /* Write zstd data... */
#ifdef ZSTD
  else if (ctl->met_type == 4)
    compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
                  out);
#endif

  /* Unknown method... */
  else {
    ERRMSG("MET_TYPE not supported!");
    LOG(3, "%d %g", precision, tolerance);
  }

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

Here is the call graph for this function:

5.21.2.79 write_prof() void write_prof (

```
const char * filename,
ctl_t * ctl,
met_t * met0,
met_t * met1,
atm_t * atm,
double t )
```

Write profile data.

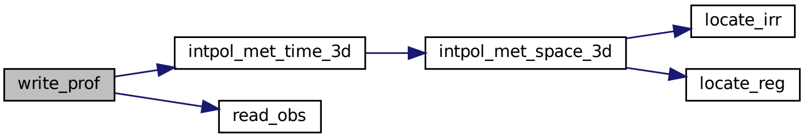
Definition at line 6363 of file libtrac.c.

```
            {

  static FILE *out;

  static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
    dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;

  static int nobs, *obscount, ip, okay;

  /* Set timer... */
  SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);

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

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

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

    /* Allocate... */
    ALLOC(lon, double,
          ctl->prof_nx);
    ALLOC(lat, double,
          ctl->prof_ny);
    ALLOC(area, double,
          ctl->prof_ny);
    ALLOC(z, double,
          ctl->prof_nz);
    ALLOC(press, double,
          ctl->prof_nz);
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
```

```
    read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create new output file... */
    LOG(1, "Write profile data: %s", 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"
            "# $11 = number of observations\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 vertical coordinates... */
    for (int iz = 0; iz < ctl->prof_nz; iz++) {
      z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
      press[iz] = P(z[iz]);
    }

    /* Set horizontal coordinates... */
    for (int ix = 0; ix < ctl->prof_nx; ix++)
      lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
    for (int iy = 0; iy < ctl->prof_ny; iy++) {
      lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
      area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
        * cos(lat[iy] * M_PI / 180.);
    }
  }

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

  /* Allocate... */
  ALLOC(mass, double,
        ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
  ALLOC(obsmean, double,
        ctl->prof_nx * ctl->prof_ny);
  ALLOC(obscount, int,
        ctl->prof_nx * ctl->prof_ny);

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Check observation data... */
    if (!isfinite(robs[i]))
      continue;

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

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

    /* Get mean observation index... */
    int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
    obsmean[idx] += robs[i];
    obscount[idx]++;
  }

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

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

```

    /* Get indices... */
    int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
    int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
    int 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... */
    int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
    mass[idx] += atm->q[ctl->qnt_m][ip];
  }

  /* Extract profiles... */
  for (int ix = 0; ix < ctl->prof_nx; ix++)
    for (int iy = 0; iy < ctl->prof_ny; iy++) {
      int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
      if (obscount[idx2] > 0) {

        /* Check profile... */
        okay = 0;
        for (int iz = 0; iz < ctl->prof_nz; iz++) {
          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
          if (mass[idx3] > 0) {
            okay = 1;
            break;
          }
        }
        if (!okay)
          continue;

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

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

          /* Get temperature, water vapor, and ozone... */
          INTPOL_INIT;
          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
                             lon[ix], lat[iy], &temp, ci, cw, 1);
          intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
                             lon[ix], lat[iy], &h2o, ci, cw, 0);
          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
                             lon[ix], lat[iy], &o3, ci, cw, 0);

          /* Calculate volume mixing ratio... */
          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
          vmr = MA / ctl->molmass * mass[idx3]
            / (RHO(press[iz], temp) * area[iy] * dz * 1e9);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %d\n",
                  t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
                  obsmean[idx2] / obscount[idx2], obscount[idx2]);
        }
      }
    }

  /* Free... */
  free(mass);
  free(obsmean);
  free(obscount);

  /* Finalize... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(lon);
    free(lat);
    free(area);
    free(z);
    free(press);
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}
```

Here is the call graph for this function:



5.21.2.80 write_sample()

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

Write sample data.

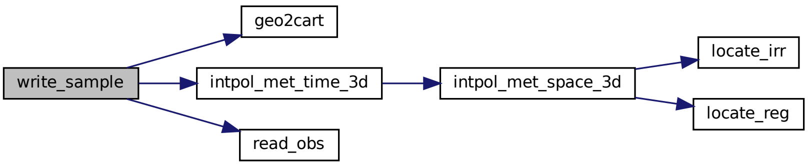
Definition at line 6591 of file libtrac.c.

```
                  {

  static FILE *out;

  static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;

  static int nobs;

  /* Set timer... */
  SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);

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

    /* Allocate... */
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create output file... */
    LOG(1, "Write sample data: %s", 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 = surface area [km^2]\n"
            "# $6 = layer depth [km]\n"
            "# $7 = number of particles [1]\n"
            "# $8 = column density [kg/m^2]\n"
            "# $9 = volume mixing ratio [ppv]\n"
            "# $10 = observed BT index [K]\n\n");

```

```
    /* Set latitude range, squared radius, and area... */
    dlat = DY2DEG(ctl->sample_dx);
    rmax2 = SQR(ctl->sample_dx);
    area = M_PI * rmax2;
  }

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

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Calculate Cartesian coordinates... */
    double x0[3];
    geo2cart(0, rlon[i], rlat[i], x0);

    /* Set pressure range... */
    double rp = P(rz[i]);
    double ptop = P(rz[i] + ctl->sample_dz);
    double pbot = P(rz[i] - ctl->sample_dz);

    /* Init... */
    double mass = 0;
    int np = 0;

    /* Loop over air parcels... */
#pragma omp parallel for default(shared) reduction(+:mass,np)
    for (int ip = 0; ip < atm->np; ip++) {

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

      /* Check latitude... */
      if (fabs(rlat[i] - atm->lat[ip]) > dlat)
        continue;

      /* Check horizontal distance... */
      double x1[3];
      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
      if (DIST2(x0, x1) > rmax2)
        continue;

      /* Check pressure... */
      if (ctl->sample_dz > 0)
        if (atm->p[ip] > pbot || atm->p[ip] < ptop)
          continue;

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

    /* Calculate column density... */
    double cd = mass / (1e6 * area);

    /* Calculate volume mixing ratio... */
    double vmr = 0;
    if (ctl->molmass > 0 && ctl->sample_dz > 0) {
      if (mass > 0) {

        /* Get temperature... */
        double temp;
        INTPOL_INIT;
        intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
                           rlon[i], rlat[i], &temp, ci, cw, 1);

        /* Calculate volume mixing ratio... */
        vmr = MA / ctl->molmass * mass
          / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
      }
    } else
      vmr = GSL_NAN;

    /* Write output... */
    fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
            rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
  }
```

```
  /* Finalize...... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}
```

Here is the call graph for this function:



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

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

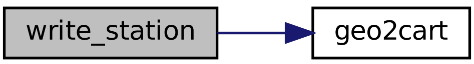
Write station data.

Definition at line 6748 of file libtrac.c.

```
                {

  static FILE *out;

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

  /* Set timer... */
  SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);

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

    /* Write info... */
    LOG(1, "Write station data: %s", 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... */
  double t0 = t - 0.5 * ctl->dt_mod;
  double 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 time range for station output... */
    if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_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.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) 2013-2023 Forschungszentrum Juelich GmbH
```

```
*/

#include "libtrac.h"

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

double buoyancy_frequency(
  double p0,
  double t0,
  double p1,
  double t1) {

  double theta0 = THETA(p0, t0);
  double theta1 = THETA(p1, t1);
  double dz = RI / MA / G0 * 0.5 * (t0 + t1) * (log(p0) - log(p1));

  return sqrt(2. * G0 / (theta0 + theta1) * (theta1 - theta0) / dz);
}

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

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;
}

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

double clim_hno3(
  clim_t * clim,
  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_p[0])
    p = clim->hno3_p[0];
  else if (p > clim->hno3_p[clim->hno3_np - 1])
    p = clim->hno3_p[clim->hno3_np - 1];

  /* Check latitude... */
  if (lat < clim->hno3_lat[0])
    lat = clim->hno3_lat[0];
  else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
    lat = clim->hno3_lat[clim->hno3_nlat - 1];

  /* Get indices... */
  int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
  int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
  int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);

  /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
  double aux00 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec][ilat][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec][ilat + 1][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec + 1][ilat][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec + 1][ilat + 1][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim->hno3_lat[ilat], aux00,
              clim->hno3_lat[ilat + 1], aux01, lat);
  aux11 = LIN(clim->hno3_lat[ilat], aux10,
              clim->hno3_lat[ilat + 1], aux11, lat);
  aux00 = LIN(clim->hno3_time[isec], aux00,
              clim->hno3_time[isec + 1], aux11, sec);

```

```
  /* Convert from ppb to ppv... */
  return GSL_MAX(1e-9 * aux00, 0.0);
}

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

void clim_hno3_init(
  clim_t * clim) {

  /* Write info... */
  LOG(1, "Initialize HNO3 data...");

  clim->hno3_ntime = 12;
  double hno3_time[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
  };
  memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));

  clim->hno3_nlat = 18;
  double hno3_lat[18] = {
    -85, -75, -65, -55, -45, -35, -25, -15, -5,
    5, 15, 25, 35, 45, 55, 65, 75, 85
  };
  memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));

  clim->hno3_np = 10;
  double hno3_p[10] = {
    4.64159, 6.81292, 10, 14.678, 21.5443,
    31.6228, 46.4159, 68.1292, 100, 146.78
  };
  memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));

  double hno3[12][18][10] = {
    {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
     {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
     {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
     {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
     {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
     {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
     {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
     {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
     {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
     {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
     {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
     {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
     {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
     {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
     {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
     {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
     {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
     {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
    {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
     {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
     {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
     {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
     {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
     {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
     {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
     {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
     {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
     {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
     {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
     {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
     {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
     {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
     {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
     {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
     {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
     {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
    {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
     {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
     {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
     {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
     {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
     {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
     {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
     {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
     {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
     {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
     {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
     {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
     {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
     {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
     {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
```

```
 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
{{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
{{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
{{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
{{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
{{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
```

```
    {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
    {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
    {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
    {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
    {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
    {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
   {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
    {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
    {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
    {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
    {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
    {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
    {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
    {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
    {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
    {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
    {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
    {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
    {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
    {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
    {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
    {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
    {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
    {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
   {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
    {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
    {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
    {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
    {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
    {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
    {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
    {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
    {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
    {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
    {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
    {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
    {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
    {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
    {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
    {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
    {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
    {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
   {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
    {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
    {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
    {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
    {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
    {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
    {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
    {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
    {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
    {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
    {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
    {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
    {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
    {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
    {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
    {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
    {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
    {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
   {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
    {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
    {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
    {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
    {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
    {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
    {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
    {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
    {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
    {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
    {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
    {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
    {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
    {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
    {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
    {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
    {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
    {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
  };
  memcpy(clim->hno3, hno3, sizeof(clim->hno3));

  /* Get range... */
  double hno3min = 1e99, hno3max = -1e99;
  for (int it = 0; it < clim->hno3_ntime; it++)
    for (int iz = 0; iz < clim->hno3_np; iz++)
      for (int iy = 0; iy < clim->hno3_nlat; iy++) {
        hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
```

```
        hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
      }

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->hno3_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->hno3_time[0], clim->hno3_time[1],
      clim->hno3_time[clim->hno3_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->hno3_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
      Z(clim->hno3_p[clim->hno3_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
      clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->hno3_lat[0], clim->hno3_lat[1],
      clim->hno3_lat[clim->hno3_nlat - 1]);
  LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
      1e-9 * hno3max);
}

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

double clim_oh(
  clim_t * clim,
  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_p[clim->oh_np - 1])
    p = clim->oh_p[clim->oh_np - 1];
  else if (p > clim->oh_p[0])
    p = clim->oh_p[0];

  /* Check latitude... */
  if (lat < clim->oh_lat[0])
    lat = clim->oh_lat[0];
  else if (lat > clim->oh_lat[clim->oh_nlat - 1])
    lat = clim->oh_lat[clim->oh_nlat - 1];

  /* Get indices... */
  int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
  int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
  int ip = locate_irr(clim->oh_p, clim->oh_np, p);

  /* Interpolate OH climatology... */
  double aux00 = LIN(clim->oh_p[ip],
                     clim->oh[isec][ip][ilat],
                     clim->oh_p[ip + 1],
                     clim->oh[isec][ip + 1][ilat], p);
  double aux01 = LIN(clim->oh_p[ip],
                     clim->oh[isec][ip][ilat + 1],
                     clim->oh_p[ip + 1],
                     clim->oh[isec][ip + 1][ilat + 1], p);
  double aux10 = LIN(clim->oh_p[ip],
                     clim->oh[isec + 1][ip][ilat],
                     clim->oh_p[ip + 1],
                     clim->oh[isec + 1][ip + 1][ilat], p);
  double aux11 = LIN(clim->oh_p[ip],
                     clim->oh[isec + 1][ip][ilat + 1],
                     clim->oh_p[ip + 1],
                     clim->oh[isec + 1][ip + 1][ilat + 1], p);
  aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
  aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
  aux00 =
    LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);

  return GSL_MAX(aux00, 0.0);
}

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

double clim_oh_diurnal(
  ctl_t * ctl,
  clim_t * clim,
  double t,
  double p,
  double lon,
  double lat) {

```

```
  double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);

  if (sza2 <= M_PI / 2. * 89. / 90.)
    return oh * exp(-ctl->oh_chem_beta / cos(sza2));
  else
    return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
}

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

void clim_oh_init(
  ctl_t * ctl,
  clim_t * clim) {

  int nt, ncid, varid;

  double *help, ohmin = 1e99, ohmax = -1e99;

  /* Write info... */
  LOG(1, "Read OH data: %s", ctl->clim_oh_filename);

  /* Open netCDF file... */
  if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
    WARN("OH climatology data are missing!");
    return;
  }

  /* Read pressure data... */
  NC_INQ_DIM("press", &clim->oh_np, 2, CP);
  NC_GET_DOUBLE("press", clim->oh_p, 1);

  /* Check ordering of pressure data... */
  if (clim->oh_p[0] < clim->oh_p[1])
    ERRMSG("Pressure data are not descending!");

  /* Read latitudes... */
  NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
  NC_GET_DOUBLE("lat", clim->oh_lat, 1);

  /* Check ordering of latitudes... */
  if (clim->oh_lat[0] > clim->oh_lat[1])
    ERRMSG("Latitude data are not ascending!");

  /* Set time data for monthly means... */
  clim->oh_ntime = 12;
  clim->oh_time[0] = 1209600.00;
  clim->oh_time[1] = 3888000.00;
  clim->oh_time[2] = 6393600.00;
  clim->oh_time[3] = 9072000.00;
  clim->oh_time[4] = 11664000.00;
  clim->oh_time[5] = 14342400.00;
  clim->oh_time[6] = 16934400.00;
  clim->oh_time[7] = 19612800.00;
  clim->oh_time[8] = 22291200.00;
  clim->oh_time[9] = 24883200.00;
  clim->oh_time[10] = 27561600.00;
  clim->oh_time[11] = 30153600.00;

  /* Check number of timesteps... */
  NC_INQ_DIM("time", &nt, 12, 12);

  /* Read OH data... */
  ALLOC(help, double,
        clim->oh_nlat * clim->oh_np * clim->oh_ntime);
  NC_GET_DOUBLE("OH", help, 1);
  for (int it = 0; it < clim->oh_ntime; it++)
    for (int iz = 0; iz < clim->oh_np; iz++)
      for (int iy = 0; iy < clim->oh_nlat; iy++) {
        clim->oh[it][iz][iy] =
          help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
          / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
                              clim->oh_lat[iy]);
        ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
        ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
      }
  free(help);

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

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->oh_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->oh_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
```

```
  LOG(2, "Pressure levels: %g, %g ... %g hPa",
      clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->oh_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
  LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
}

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

double clim_oh_init_help(
  double beta,
  double time,
  double lat) {

  double aux, lon, sum = 0;

  int n = 0;

  /* Integrate day/night correction factor over longitude... */
  for (lon = -180; lon < 180; lon += 1) {
    aux = sza(time, lon, lat);
    if (aux <= M_PI / 2. * 85. / 90.)
      sum += exp(-beta / cos(aux));
    else
      sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
    n++;
  }
  return sum / (double) n;
}

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

double clim_h2o2(
  clim_t * clim,
  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->h2o2_p[clim->h2o2_np - 1])
    p = clim->h2o2_p[clim->h2o2_np - 1];
  else if (p > clim->h2o2_p[0])
    p = clim->h2o2_p[0];

  /* Check latitude... */
  if (lat < clim->h2o2_lat[0])
    lat = clim->h2o2_lat[0];
  else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
    lat = clim->h2o2_lat[clim->h2o2_nlat - 1];

  /* Get indices... */
  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);

  /* Interpolate H2O2 climatology... */
  double aux00 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec][ip][ilat],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec][ip + 1][ilat], p);
  double aux01 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec][ip][ilat + 1],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec][ip + 1][ilat + 1], p);
  double aux10 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec + 1][ip][ilat],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec + 1][ip + 1][ilat], p);
  double aux11 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec + 1][ip][ilat + 1],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
  aux00 =
    LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
  aux11 =
    LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
  aux00 =
    LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);

  return GSL_MAX(aux00, 0.0);
}
```

```

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

void clim_h2o2_init(
  ctl_t * ctl,
  clim_t * clim) {

  int ncid, varid, it, iy, iz, nt;

  double *help, h2o2min = 1e99, h2o2max = -1e99;

  /* Write info... */
  LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);

  /* Open netCDF file... */
  if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
    WARN("H2O2 climatology data are missing!");
    return;
  }

  /* Read pressure data... */
  NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
  NC_GET_DOUBLE("press", clim->h2o2_p, 1);

  /* Check ordering of pressure data... */
  if (clim->h2o2_p[0] < clim->h2o2_p[1])
    ERRMSG("Pressure data are not descending!");

  /* Read latitudes... */
  NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
  NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);

  /* Check ordering of latitude data... */
  if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
    ERRMSG("Latitude data are not ascending!");

  /* Set time data (for monthly means)... */
  clim->h2o2_ntime = 12;
  clim->h2o2_time[0] = 1209600.00;
  clim->h2o2_time[1] = 3888000.00;
  clim->h2o2_time[2] = 6393600.00;
  clim->h2o2_time[3] = 9072000.00;
  clim->h2o2_time[4] = 11664000.00;
  clim->h2o2_time[5] = 14342400.00;
  clim->h2o2_time[6] = 16934400.00;
  clim->h2o2_time[7] = 19612800.00;
  clim->h2o2_time[8] = 22291200.00;
  clim->h2o2_time[9] = 24883200.00;
  clim->h2o2_time[10] = 27561600.00;
  clim->h2o2_time[11] = 30153600.00;

  /* Check number of timesteps... */
  NC_INQ_DIM("time", &nt, 12, 12);

  /* Read data... */
  ALLOC(help, double,
        clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
  NC_GET_DOUBLE("h2o2", help, 1);
  for (it = 0; it < clim->h2o2_ntime; it++)
    for (iz = 0; iz < clim->h2o2_np; iz++)
      for (iy = 0; iy < clim->h2o2_nlat; iy++) {
        clim->h2o2[it][iz][iy] =
          help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
        h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
        h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
      }
  free(help);

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

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->h2o2_time[0], clim->h2o2_time[1],
      clim->h2o2_time[clim->h2o2_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
      Z(clim->h2o2_p[clim->h2o2_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
      clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->h2o2_lat[0], clim->h2o2_lat[1],
      clim->h2o2_lat[clim->h2o2_nlat - 1]);
  LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
```

```
}

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

double clim_tropo(
  clim_t * clim,
  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_time, clim->tropo_ntime, sec);
  int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);

  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
  double p0 = LIN(clim->tropo_lat[ilat],
                  clim->tropo[isec][ilat],
                  clim->tropo_lat[ilat + 1],
                  clim->tropo[isec][ilat + 1], lat);
  double p1 = LIN(clim->tropo_lat[ilat],
                  clim->tropo[isec + 1][ilat],
                  clim->tropo_lat[ilat + 1],
                  clim->tropo[isec + 1][ilat + 1], lat);
  return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
}

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

void clim_tropo_init(
  clim_t * clim) {

  /* Write info... */
  LOG(1, "Initialize tropopause data...");

  clim->tropo_ntime = 12;
  double tropo_time[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
  };
  memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));

  clim->tropo_nlat = 73;
  double tropo_lat[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
  };
  memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));

  double tropo[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}
  };
  memcpy(clim->tropo, tropo, sizeof(clim->tropo));

  /* Get range... */
  double tropomin = 1e99, tropomax = -1e99;
  for (int it = 0; it < clim->tropo_ntime; it++)
    for (int iy = 0; iy < clim->tropo_nlat; iy++) {
      tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
      tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
    }

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->tropo_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->tropo_time[0], clim->tropo_time[1],
      clim->tropo_time[clim->tropo_ntime - 1]);
  LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
```

```
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->tropo_lat[0], clim->tropo_lat[1],
      clim->tropo_lat[clim->tropo_nlat - 1]);
  LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
      Z(tropomin));
  LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
}

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

void compress_pack(
  char *varname,
  float *array,
  size_t nxy,
  size_t nz,
  int decompress,
  FILE * inout) {

  double min[EP], max[EP], off[EP], scl[EP];

  unsigned short *sarray;

  /* Allocate... */
  ALLOC(sarray, unsigned short,
        nxy * nz);

  /* Read compressed stream and decompress array... */
  if (decompress) {

    /* Write info... */
    LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
        varname, 100. * sizeof(unsigned short) / sizeof(float));

    /* Read data... */
    FREAD(&scl, double,
          nz,
          inout);
    FREAD(&off, double,
          nz,
          inout);
    FREAD(sarray, unsigned short,
          nxy * nz,
          inout);

    /* Convert to float... */
#pragma omp parallel for default(shared)
    for (size_t ixy = 0; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++)
        array[ixy * nz + iz]
          = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
  }

  /* Compress array and output compressed stream... */
  else {

    /* Write info... */
    LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
        varname, 100. * sizeof(unsigned short) / sizeof(float));

    /* Get range... */
    for (size_t iz = 0; iz < nz; iz++) {
      min[iz] = array[iz];
      max[iz] = array[iz];
    }
    for (size_t ixy = 1; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++) {
        if (array[ixy * nz + iz] < min[iz])
          min[iz] = array[ixy * nz + iz];
        if (array[ixy * nz + iz] > max[iz])
          max[iz] = array[ixy * nz + iz];
      }

    /* Get offset and scaling factor... */
    for (size_t iz = 0; iz < nz; iz++) {
      scl[iz] = (max[iz] - min[iz]) / 65533.;
      off[iz] = min[iz];
    }

    /* Convert to short... */
#pragma omp parallel for default(shared)
    for (size_t ixy = 0; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++)
        if (scl[iz] != 0)
          sarray[ixy * nz + iz] = (unsigned short)
            ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
        else
          sarray[ixy * nz + iz] = 0;
```

```

    /* Write data... */
    FWRITE(&scl, double,
           nz,
           inout);
    FWRITE(&off, double,
           nz,
           inout);
    FWRITE(sarray, unsigned short,
           nxy * nz,
           inout);
  }

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

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

#ifdef ZFP
void compress_zfp(
  char *varname,
  float *array,
  int nx,
  int ny,
  int nz,
  int precision,
  double tolerance,
  int decompress,
  FILE * inout) {

  zfp_type type;                /* array scalar type */
  zfp_field *field;             /* array meta data */
  zfp_stream *zfp;              /* compressed stream */
  void *buffer;                 /* storage for compressed stream */
  size_t bufsize;               /* byte size of compressed buffer */
  bitstream *stream;            /* bit stream to write to or read from */
  size_t zfpsize;               /* byte size of compressed stream */

  /* Allocate meta data for the 3D array a[nz][ny][nx]... */
  type = zfp_type_float;
  field = zfp_field_3d(array, type, (uint) nx, (uint) ny, (uint) nz);

  /* Allocate meta data for a compressed stream... */
  zfp = zfp_stream_open(NULL);

  /* Set compression mode... */
  int actual_prec = 0;
  double actual_tol = 0;
  if (precision > 0)
    actual_prec = (int) zfp_stream_set_precision(zfp, (uint) precision);
  else if (tolerance > 0)
    actual_tol = zfp_stream_set_accuracy(zfp, tolerance);
  else
    ERRMSG("Set precision or tolerance!");

  /* Allocate buffer for compressed data... */
  bufsize = zfp_stream_maximum_size(zfp, field);
  buffer = malloc(bufsize);

  /* Associate bit stream with allocated buffer... */
  stream = stream_open(buffer, bufsize);
  zfp_stream_set_bit_stream(zfp, stream);
  zfp_stream_rewind(zfp);

  /* Read compressed stream and decompress array... */
  if (decompress) {
    FREAD(&zfpsize, size_t,
          1,
          inout);
    if (fread(buffer, 1, zfpsize, inout) != zfpsize)
      ERRMSG("Error while reading zfp data!");
    if (!zfp_decompress(zfp, field)) {
      ERRMSG("Decompression failed!");
    }
    LOG(2, "Read 3-D variable: %s "
        "(zfp, PREC= %d, TOL= %g, RATIO= %g %%)",
        varname, actual_prec, actual_tol,
        (100. * (double) zfpsize) / (double) (nx * ny * nz));
  }

  /* Compress array and output compressed stream... */
  else {
    zfpsize = zfp_compress(zfp, field);
    if (!zfpsize) {
      ERRMSG("Compression failed!");
    } else {
```

```
      FWRITE(&zfpsize, size_t,
             1,
             inout);
      if (fwrite(buffer, 1, zfpsize, inout) != zfpsize)
        ERRMSG("Error while writing zfp data!");
    }
    LOG(2, "Write 3-D variable: %s "
        "(zfp, PREC= %d, TOL= %g, RATIO= %g %%)",
        varname, actual_prec, actual_tol,
        (100. * (double) zfpsize) / (double) (nx * ny * nz));
  }

  /* Free... */
  zfp_field_free(field);
  zfp_stream_close(zfp);
  stream_close(stream);
  free(buffer);
}
#endif

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

#ifdef ZSTD
void compress_zstd(
  char *varname,
  float *array,
  size_t n,
  int decompress,
  FILE * inout) {

  /* Get buffer sizes... */
  size_t uncomprLen = n * sizeof(float);
  size_t comprLen = ZSTD_compressBound(uncomprLen);
  size_t compsize;

  /* Allocate... */
  char *compr = (char *) calloc((uint) comprLen, 1);
  char *uncompr = (char *) array;

  /* Read compressed stream and decompress array... */
  if (decompress) {
    FREAD(&comprLen, size_t,
          1,
          inout);
    if (fread(compr, 1, comprLen, inout) != comprLen)
      ERRMSG("Error while reading zstd data!");
    compsize = ZSTD_decompress(uncompr, uncomprLen, compr, comprLen);
    if (ZSTD_isError(compsize)) {
      ERRMSG("Decompression failed!");
    }
    LOG(2, "Read 3-D variable: %s (zstd, RATIO= %g %%)",
        varname, (100. * (double) comprLen) / (double) uncomprLen);
  }

  /* Compress array and output compressed stream... */
  else {
    compsize = ZSTD_compress(compr, comprLen, uncompr, uncomprLen, 0);
    if (ZSTD_isError(compsize)) {
      ERRMSG("Compression failed!");
    } else {
      FWRITE(&compsize, size_t,
             1,
             inout);
      if (fwrite(compr, 1, compsize, inout) != compsize)
        ERRMSG("Error while writing zstd data!");
    }
    LOG(2, "Write 3-D variable: %s (zstd, RATIO= %g %%)",
        varname, (100. * (double) compsize) / (double) uncomprLen);
  }

  /* Free... */
  free(compr);
}
#endif

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

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

  const int
    d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
    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 = d0l[mon - 1] + day - 1;
  else
    *doy = d0[mon - 1] + day - 1;
}

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

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

  const int
    d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
    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,
  clim_t * clim,
  double t,
  met_t ** met0,
  met_t ** met1) {

  static int init;

  met_t *mets;

  char cachefile[LEN], cmd[2 * LEN], filename[LEN];

  /* Set timer... */
  SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);

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

    /* Read meteo data... */
    get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
                 ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met0))
      ERRMSG("Cannot open file!");

    get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
                 ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met1))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met0up = *met0;
```

```
    met_t *met1up = *met1;
#ifdef ASYNCIO
#pragma acc update device(met0up[:1],met1up[:1]) async(5)
#else
#pragma acc update device(met0up[:1],met1up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
                   ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }

  /* Read new data for forward trajectories... */
  if (t > (*met1)->time) {

    /* Pointer swap... */
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;

    /* Read new meteo data... */
    get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met1))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met1up = *met1;
#ifdef ASYNCIO
#pragma acc update device(met1up[:1]) async(5)
#else
#pragma acc update device(met1up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
                   cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }
  /* Read new data for backward trajectories... */
  if (t < (*met0)->time) {

    /* Pointer swap... */
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;

    /* Read new meteo data... */
    get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met0))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met0up = *met0;
#ifdef ASYNCIO
#pragma acc update device(met0up[:1]) async(5)
#else
#pragma acc update device(met0up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
                   cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }
  /* Check that grids are consistent... */
  if ((*met0)->nx != 0 && (*met1)->nx != 0) {
```

```
    if ((*met0)->nx != (*met1)->nx
        || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
      ERRMSG("Meteo grid dimensions do not match!");
    for (int ix = 0; ix < (*met0)->nx; ix++)
      if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
        ERRMSG("Meteo grid longitudes do not match!");
    for (int iy = 0; iy < (*met0)->ny; iy++)
      if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
        ERRMSG("Meteo grid latitudes do not match!");
    for (int ip = 0; ip < (*met0)->np; ip++)
      if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
        ERRMSG("Meteo grid pressure levels do not match!");
  }
}

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

void get_met_help(
  ctl_t * ctl,
  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 of MPTRAC meteo files... */
  if (ctl->clams_met_data == 0) {
    if (ctl->met_type == 0)
      sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
    else if (ctl->met_type == 1)
      sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
    else if (ctl->met_type == 2)
      sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
    else if (ctl->met_type == 3)
      sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
    else if (ctl->met_type == 4)
      sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", 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);
  }

  /* Set filename of CLaMS meteo files... */
  else {
    sprintf(filename, "%s_YYMMDDHH.nc", metbase);
    sprintf(repl, "%d", year);
    get_met_replace(filename, "YYYY", repl);
    sprintf(repl, "%d", year % 100);
    get_met_replace(filename, "YY", 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];

```

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

    /* Replace sub-string... */
    char *ch;
    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) {

  /* Initialize interpolation... */
  if (init) {

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

    /* Get interpolation indices... */
    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);

    /* Get interpolation weights... */
    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) {

  /* Initialize interpolation... */
  if (init) {

```

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

    /* Get interpolation indices... */
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);

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

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

#ifdef UVW
void intpol_met_space_uvw(
  met_t * met,
  double p,
  double lon,
  double lat,
  double *u,
  double *v,
  double *w,
  int *ci,
  double *cw,
  int init) {

  /* Initialize interpolation... */
  if (init) {

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

    /* Get interpolation indices... */
    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);

    /* Get interpolation weights... */
    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 u00 =
    cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][0] -
             met->uvw[ci[1]][ci[2]][ci[0] + 1][0])
    + met->uvw[ci[1]][ci[2]][ci[0] + 1][0];
  double u01 =
    cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][0] -
             met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][0])
    + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][0];
```

```
  double u10 =
    cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][0] -
             met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][0])
    + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][0];
  double u11 =
    cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][0] -
             met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][0])
    + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][0];

  double v00 =
    cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][1] -
             met->uvw[ci[1]][ci[2]][ci[0] + 1][1])
    + met->uvw[ci[1]][ci[2]][ci[0] + 1][1];
  double v01 =
    cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][1] -
             met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][1])
    + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][1];
  double v10 =
    cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][1] -
             met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][1])
    + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][1];
  double v11 =
    cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][1] -
             met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][1])
    + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][1];

  double w00 =
    cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][2] -
             met->uvw[ci[1]][ci[2]][ci[0] + 1][2])
    + met->uvw[ci[1]][ci[2]][ci[0] + 1][2];
  double w01 =
    cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][2] -
             met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][2])
    + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][2];
  double w10 =
    cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][2] -
             met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][2])
    + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][2];
  double w11 =
    cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][2] -
             met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][2])
    + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][2];

  /* Interpolate horizontally... */
  u00 = cw[2] * (u00 - u01) + u01;
  u11 = cw[2] * (u10 - u11) + u11;
  *u = cw[1] * (u00 - u11) + u11;

  v00 = cw[2] * (v00 - v01) + v01;
  v11 = cw[2] * (v10 - v11) + v11;
  *v = cw[1] * (v00 - v11) + v11;

  w00 = cw[2] * (w00 - w01) + w01;
  w11 = cw[2] * (w10 - w11) + w11;
  *w = cw[1] * (w00 - w11) + w11;
}
#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) {

  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, 0);

  /* 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, 0);

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

  /* Interpolate... */
  if (isfinite(var0) && isfinite(var1))
    *var = wt * (var0 - var1) + var1;
  else if (wt < 0.5)
    *var = var1;
  else
    *var = var0;
}

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

#ifdef UVW
void intpol_met_time_uvw(
  met_t * met0,
  met_t * met1,
  double ts,
  double p,
  double lon,
  double lat,
  double *u,
  double *v,
  double *w) {

  double u0, u1, v0, v1, w0, w1, wt;

  /* Spatial interpolation... */
  INTPOL_INIT;
  intpol_met_space_uvw(met0, p, lon, lat, &u0, &v0, &w0, ci, cw, 1);
  intpol_met_space_uvw(met1, p, lon, lat, &u1, &v1, &w1, ci, cw, 0);

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

  /* Interpolate... */
  *u = wt * (u0 - u1) + u1;
  *v = wt * (v0 - v1) + v1;
  *w = wt * (w0 - w1) + w1;
}
#endif

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

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);
}

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

double lapse_rate(
  double t,
  double h2o) {

  /*
     Calculate moist adiabatic lapse rate [K/km] from temperature [K]
     and water vapor volume mixing ratio [1].

     Reference: https://en.wikipedia.org/wiki/Lapse_rate
   */

  const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));

  return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
}

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

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)
    return 0;
  else if (i > n - 2)
    return n - 2;
  else
    return i;
}

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

double nat_temperature(
  double p,
  double h2o,
  double hno3) {

  /* Check water vapor vmr... */
  h2o = GSL_MAX(h2o, 0.1e-6);

```

```
  /* Calculate T_NAT... */
  double p_hno3 = hno3 * p / 1.333224;
  double p_h2o = h2o * p / 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 tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
  if (x2 > 0)
    tnat = x2;

  return tnat;
}

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

void quicksort(
  double arr[],
  int brr[],
  int low,
  int high) {

  if (low < high) {
    int pi = quicksort_partition(arr, brr, low, high);

#pragma omp task firstprivate(arr,brr,low,pi)
    {
      quicksort(arr, brr, low, pi - 1);
    }

    // #pragma omp task firstprivate(arr,brr,high,pi)
    {
      quicksort(arr, brr, pi + 1, high);
    }
  }
}

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

int quicksort_partition(
  double arr[],
  int brr[],
  int low,
  int high) {

  double pivot = arr[high];
  int i = (low - 1);

  for (int j = low; j <= high - 1; j++)
    if (arr[j] <= pivot) {
      i++;
      SWAP(arr[i], arr[j], double);
      SWAP(brr[i], brr[j], int);
    }
  SWAP(arr[high], arr[i + 1], double);
  SWAP(brr[high], brr[i + 1], int);

  return (i + 1);
}

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

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

  int result;

  /* Set timer... */
  SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);

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

  /* Write info... */
  LOG(1, "Read atmospheric data: %s", filename);

  /* Read ASCII data... */
  if (ctl->atm_type == 0)
    result = read_atm_asc(filename, ctl, atm);

  /* Read binary data... */
  else if (ctl->atm_type == 1)
    result = read_atm_bin(filename, ctl, atm);

  /* Read netCDF data... */
```

```
  else if (ctl->atm_type == 2)
    result = read_atm_nc(filename, ctl, atm);

  /* Read CLaMS data... */
  else if (ctl->atm_type == 3)
    result = read_atm_clams(filename, ctl, atm);

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

  /* Check result... */
  if (result != 1)
    return 0;

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of particles: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  for (int iq = 0; iq < ctl->nq; iq++) {
    char msg[LEN];
    sprintf(msg, "Quantity %s range: %s ... %s %s",
            ctl->qnt_name[iq], ctl->qnt_format[iq],
            ctl->qnt_format[iq], ctl->qnt_unit[iq]);
    gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
    LOG(2, msg, mini, maxi);
  }

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

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

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

  FILE *in;

  /* Open file... */
  if (!(in = fopen(filename, "r"))) {
    WARN("Cannot open file!");
    return 0;
  }

  /* Read line... */
  char line[LEN];
  while (fgets(line, LEN, in)) {

    /* Read data... */
    char *tok;
    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 (int 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);

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

```

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

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

  FILE *in;

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

  /* Check version of binary data... */
  int version;
  FREAD(&version, int,
        1,
        in);
  if (version != 100)
    ERRMSG("Wrong version of binary data!");

  /* 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 (int iq = 0; iq < ctl->nq; iq++)
    FREAD(atm->q[iq], double,
            (size_t) atm->np,
          in);

  /* Read final flag... */
  int final;
  FREAD(&final, int,
        1,
        in);
  if (final != 999)
    ERRMSG("Error while reading binary data!");

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

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

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

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

  int ncid, varid;

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

  /* Get dimensions... */
  NC_INQ_DIM("NPARTS", &atm->np, 1, NP);

  /* Get time... */
  if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
    NC(nc_get_var_double(ncid, varid, atm->time));
  } else {
    WARN("TIME_INIT not found use time instead!");
    double time_init;
    NC_GET_DOUBLE("time", &time_init, 1);
    for (int ip = 0; ip < atm->np; ip++) {
      atm->time[ip] = time_init;
    }
  }

  /* Read zeta coordinate, pressure is optional... */
  if (ctl->vert_coord_ap == 1) {
```

```
    NC_GET_DOUBLE("ZETA", atm->zeta, 1);
    NC_GET_DOUBLE("PRESS", atm->p, 0);
  }

  /* Read pressure, zeta coordinate is optional... */
  else {
    NC_GET_DOUBLE("PRESS", atm->p, 1);
    NC_GET_DOUBLE("ZETA", atm->zeta, 0);
  }

  /* Read longitude and latitude... */
  NC_GET_DOUBLE("LON", atm->lon, 1);
  NC_GET_DOUBLE("LAT", atm->lat, 1);

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

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

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

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

  int ncid, varid;

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

  /* Get dimensions... */
  NC_INQ_DIM("obs", &atm->np, 1, NP);

  /* Read geolocations... */
  NC_GET_DOUBLE("time", atm->time, 1);
  NC_GET_DOUBLE("press", atm->p, 1);
  NC_GET_DOUBLE("lon", atm->lon, 1);
  NC_GET_DOUBLE("lat", atm->lat, 1);

  /* Read variables... */
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

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

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

void read_clim(
  ctl_t * ctl,
  clim_t * clim) {

  /* Set timer... */
  SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);

  /* Init tropopause climatology... */
  clim_tropo_init(clim);

  /* Init HNO3 climatology... */
  clim_hno3_init(clim);

  /* Read OH climatology... */
  if (ctl->clim_oh_filename[0] != '-')
    clim_oh_init(ctl, clim);

  /* Read H2O2 climatology... */
  if (ctl->clim_h2o2_filename[0] != '-')
    clim_h2o2_init(ctl, clim);
}

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

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

  /* Set timer... */
```

```
  SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);

  /* Write info... */
  LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
      "(executable: %s | version: %s | compiled: %s, %s)\n",
      argv[0], VERSION, __DATE__, __TIME__);

  /* Initialize quantity indices... */
  ctl->qnt_idx = -1;
  ctl->qnt_ens = -1;
  ctl->qnt_stat = -1;
  ctl->qnt_m = -1;
  ctl->qnt_vmr = -1;
  ctl->qnt_rp = -1;
  ctl->qnt_rhop = -1;
  ctl->qnt_ps = -1;
  ctl->qnt_ts = -1;
  ctl->qnt_zs = -1;
  ctl->qnt_us = -1;
  ctl->qnt_vs = -1;
  ctl->qnt_pbl = -1;
  ctl->qnt_pt = -1;
  ctl->qnt_tt = -1;
  ctl->qnt_zt = -1;
  ctl->qnt_h2ot = -1;
  ctl->qnt_z = -1;
  ctl->qnt_p = -1;
  ctl->qnt_t = -1;
  ctl->qnt_rho = -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_pct = -1;
  ctl->qnt_pcb = -1;
  ctl->qnt_cl = -1;
  ctl->qnt_plcl = -1;
  ctl->qnt_plfc = -1;
  ctl->qnt_pel = -1;
  ctl->qnt_cape = -1;
  ctl->qnt_cin = -1;
  ctl->qnt_hno3 = -1;
  ctl->qnt_oh = -1;
  ctl->qnt_vmrimpl = -1;
  ctl->qnt_mloss_oh = -1;
  ctl->qnt_mloss_h2o2 = -1;
  ctl->qnt_mloss_wet = -1;
  ctl->qnt_mloss_dry = -1;
  ctl->qnt_mloss_decay = -1;
  ctl->qnt_psat = -1;
  ctl->qnt_psice = -1;
  ctl->qnt_pw = -1;
  ctl->qnt_sh = -1;
  ctl->qnt_rh = -1;
  ctl->qnt_rhice = -1;
  ctl->qnt_theta = -1;
  ctl->qnt_zeta = -1;
  ctl->qnt_tvirt = -1;
  ctl->qnt_lapse = -1;
  ctl->qnt_vh = -1;
  ctl->qnt_vz = -1;
  ctl->qnt_pv = -1;
  ctl->qnt_tdew = -1;
  ctl->qnt_tice = -1;
  ctl->qnt_tsts = -1;
  ctl->qnt_tnat = -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_LONGNAME", iq, ctl->qnt_name[iq],
             ctl->qnt_longname[iq]);
    scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
             ctl->qnt_format[iq]);

    /* Try to identify quantity... */
    SET_QNT(qnt_idx, "idx", "particle index", "-")
      SET_QNT(qnt_ens, "ens", "ensemble index", "-")
      SET_QNT(qnt_stat, "stat", "station flag", "-")
```

```
      SET_QNT(qnt_m, "m", "mass", "kg")
      SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
      SET_QNT(qnt_rp, "rp", "particle radius", "microns")
      SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
      SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
      SET_QNT(qnt_ts, "ts", "surface temperature", "K")
      SET_QNT(qnt_zs, "zs", "surface height", "km")
      SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
      SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
      SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
      SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
      SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
      SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
      SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
      SET_QNT(qnt_z, "z", "geopotential height", "km")
      SET_QNT(qnt_p, "p", "pressure", "hPa")
      SET_QNT(qnt_t, "t", "temperature", "K")
      SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
      SET_QNT(qnt_u, "u", "zonal wind", "m/s")
      SET_QNT(qnt_v, "v", "meridional wind", "m/s")
      SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
      SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
      SET_QNT(qnt_o3, "o3", "ozone", "ppv")
      SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
      SET_QNT(qnt_iwc, "iwc", "cloud liquid water content", "kg/kg")
      SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
      SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
      SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
      SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
      SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
      SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
      SET_QNT(qnt_cape, "cape", "convective available potential energy",
              "J/kg")
      SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
      SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
      SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
      SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
      SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
      SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
              "kg")
      SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
              "kg")
      SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
              "kg")
      SET_QNT(qnt_mloss_decay, "mloss_decay",
              "mass loss due to exponential decay", "kg")
      SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
      SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
      SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
      SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
      SET_QNT(qnt_rh, "rh", "relative humidity", "%%")
      SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%%")
      SET_QNT(qnt_theta, "theta", "potential temperature", "K")
      SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
      SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
      SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
      SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
      SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
      SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
      SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
      SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
      SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
      SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
      scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
  }

  /* netCDF I/O parameters... */
  ctl->chunkszhint =
    (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
                      NULL);
  ctl->read_mode =
    (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);

  /* Vertical coordinates and velocities... */
  ctl->vert_coord_ap =
    (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
  ctl->vert_coord_met =
    (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
  ctl->vert_vel =
    (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
  ctl->clams_met_data =
    (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);

  /* 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, "180", NULL);

  /* Meteo data... */
  scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
  ctl->met_type =
    (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
  ctl->met_nc_scale =
    (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", 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);
  if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
    ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
  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);
  if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
    ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
  ctl->met_detrend =
    scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", 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_geopot_sx
    = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
  ctl->met_geopot_sy
    = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
  ctl->met_relhum
    = (int) scan_ctl(filename, argc, argv, "MET_RELHUM", -1, "0", NULL);
  ctl->met_tropo =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
  if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
    ERRMSG("Set MET_TROPO = 0 ... 5!");
  ctl->met_tropo_lapse =
    scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
  ctl->met_tropo_nlev =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
  ctl->met_tropo_lapse_sep =
    scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
  ctl->met_tropo_nlev_sep =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
                   NULL);
  ctl->met_tropo_pv =
    scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
  ctl->met_tropo_theta =
    scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
  ctl->met_tropo_spline =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
  ctl->met_cloud =
    (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
  if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
    ERRMSG("Set MET_CLOUD = 0 ... 3!");
  ctl->met_cloud_min =
    scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
  ctl->met_dt_out =
    scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
  ctl->met_cache =
    (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);

  /* Sorting... */
  ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);

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

  /* Advection parameters... */
  ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
  if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
    ERRMSG("Set ADVECT to 1, 2, or 4!");
  ctl->reflect =
    (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);

  /* 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);

  /* Convection... */
  ctl->conv_cape
    = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
  ctl->conv_cin
    = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
  ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
  ctl->conv_mix
    = (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "1", NULL);
  ctl->conv_mix_bot
    = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
  ctl->conv_mix_top
    = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);

  /* Boundary conditions... */
  ctl->bound_mass =
    scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
  ctl->bound_mass_trend =
    scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
  ctl->bound_vmr =
    scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
  ctl->bound_vmr_trend =
    scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
  ctl->bound_lat0 =
    scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
  ctl->bound_lat1 =
    scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
  ctl->bound_p0 =
    scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
  ctl->bound_p1 =
    scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
  ctl->bound_dps =
    scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
  ctl->bound_dzs =
    scan_ctl(filename, argc, argv, "BOUND_DZS", -1, "-999", NULL);
  ctl->bound_zetas =
    scan_ctl(filename, argc, argv, "BOUND_ZETAS", -1, "-999", NULL);
  ctl->bound_pbl =
    (int) scan_ctl(filename, argc, argv, "BOUND_PBL", -1, "0", NULL);

  /* Species parameters... */
  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
  if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
    ctl->molmass = 120.907;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
  } else if (strcasecmp(ctl->species, "CFCl3") == 0) {
    ctl->molmass = 137.359;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
  } else if (strcasecmp(ctl->species, "CH4") == 0) {
    ctl->molmass = 16.043;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 2.45e-12;
    ctl->oh_chem[1] = 1775;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
  } else if (strcasecmp(ctl->species, "CO") == 0) {
    ctl->molmass = 28.01;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 6.9e-33;
    ctl->oh_chem[1] = 2.1;
    ctl->oh_chem[2] = 1.1e-12;
    ctl->oh_chem[3] = -1.3;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
  } else if (strcasecmp(ctl->species, "CO2") == 0) {
    ctl->molmass = 44.009;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
  } else if (strcasecmp(ctl->species, "H2O") == 0) {
    ctl->molmass = 18.01528;
  } else if (strcasecmp(ctl->species, "N2O") == 0) {
    ctl->molmass = 44.013;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
  } else if (strcasecmp(ctl->species, "NH3") == 0) {
    ctl->molmass = 17.031;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 1.7e-12;
    ctl->oh_chem[1] = 710;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
```

```
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
  } else if (strcasecmp(ctl->species, "HNO3") == 0) {
    ctl->molmass = 63.012;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
  } else if (strcasecmp(ctl->species, "NO") == 0) {
    ctl->molmass = 30.006;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 7.1e-31;
    ctl->oh_chem[1] = 2.6;
    ctl->oh_chem[2] = 3.6e-11;
    ctl->oh_chem[3] = 0.1;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
  } else if (strcasecmp(ctl->species, "NO2") == 0) {
    ctl->molmass = 46.005;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 1.8e-30;
    ctl->oh_chem[1] = 3.0;
    ctl->oh_chem[2] = 2.8e-11;
    ctl->oh_chem[3] = 0.0;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
  } else if (strcasecmp(ctl->species, "O3") == 0) {
    ctl->molmass = 47.997;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 1.7e-12;
    ctl->oh_chem[1] = 940;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
  } else if (strcasecmp(ctl->species, "SF6") == 0) {
    ctl->molmass = 146.048;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
  } else if (strcasecmp(ctl->species, "SO2") == 0) {
    ctl->molmass = 64.066;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 2.9e-31;
    ctl->oh_chem[1] = 4.1;
    ctl->oh_chem[2] = 1.7e-12;
    ctl->oh_chem[3] = -0.2;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
  } else {
    ctl->molmass =
      scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
    ctl->oh_chem_reaction =
      (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
    ctl->h2o2_chem_reaction =
      (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
                     NULL);
    for (int ip = 0; ip < 4; ip++)
      ctl->oh_chem[ip] =
        scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
    ctl->dry_depo_vdep =
      scan_ctl(filename, argc, argv, "DRY_DEPO_VDEP", -1, "0", NULL);
    ctl->dry_depo_dp =
      scan_ctl(filename, argc, argv, "DRY_DEPO_DP", -1, "30", NULL);
    ctl->wet_depo_ic_a =
      scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
    ctl->wet_depo_ic_b =
      scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
    ctl->wet_depo_bc_a =
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
    ctl->wet_depo_bc_b =
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
    for (int ip = 0; ip < 3; ip++)
      ctl->wet_depo_ic_h[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
    for (int ip = 0; ip < 1; ip++)
      ctl->wet_depo_bc_h[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
  }

  /* Wet deposition... */
  ctl->wet_depo_pre[0] =
    scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
  ctl->wet_depo_pre[1] =
    scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
  ctl->wet_depo_ic_ret_ratio =
    scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
  ctl->wet_depo_bc_ret_ratio =
    scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);

  /* OH chemistry... */
  ctl->oh_chem_beta =
    scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
```

```
scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
         "../../data/clams_radical_species.nc", ctl->clim_oh_filename);

/* H2O2 chemistry... */
ctl->h2o2_chem_cc =
  scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
         "../../data/cams_H2O2.nc", ctl->clim_h2o2_filename);

/* Chemistry grid... */
ctl->chemgrid_z0 =
  scan_ctl(filename, argc, argv, "CHEMGRID_Z0", -1, "0", NULL);
ctl->chemgrid_z1 =
  scan_ctl(filename, argc, argv, "CHEMGRID_Z1", -1, "100", NULL);
ctl->chemgrid_nz =
  (int) scan_ctl(filename, argc, argv, "CHEMGRID_NZ", -1, "1", NULL);
ctl->chemgrid_lon0 =
  scan_ctl(filename, argc, argv, "CHEMGRID_LON0", -1, "-180", NULL);
ctl->chemgrid_lon1 =
  scan_ctl(filename, argc, argv, "CHEMGRID_LON1", -1, "180", NULL);
ctl->chemgrid_nx =
  (int) scan_ctl(filename, argc, argv, "CHEMGRID_NX", -1, "360", NULL);
ctl->chemgrid_lat0 =
  scan_ctl(filename, argc, argv, "CHEMGRID_LAT0", -1, "-90", NULL);
ctl->chemgrid_lat1 =
  scan_ctl(filename, argc, argv, "CHEMGRID_LAT1", -1, "90", NULL);
ctl->chemgrid_ny =
  (int) scan_ctl(filename, argc, argv, "CHEMGRID_NY", -1, "180", NULL);

/* Exponential decay... */
ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
ctl->tdec_strat =
  scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);

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

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

/* 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);
  ctl->grid_type =
    (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", 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 sample data... */
  scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
           ctl->sample_basename);
  scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
           ctl->sample_obsfile);
  ctl->sample_dx =
    scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
  ctl->sample_dz =
    scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", 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);
  ctl->stat_t0 =
    scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
  ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
}

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

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

  /* Write info... */
  LOG(1, "Read meteo data: %s", filename);

  /* Read netCDF data... */
  if (ctl->met_type == 0) {

    int ncid;

    /* Open netCDF file... */
    if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
        NC_NOERR) {
      WARN("Cannot open file!");
      return 0;
    }

    /* Read coordinates of meteo data... */
    read_met_grid(filename, ncid, ctl, met);

    /* Read meteo data on vertical levels... */
```

```
    read_met_levels(ncid, ctl, met);

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

    /* Read surface data... */
    read_met_surface(ncid, met, ctl);

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

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

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

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

    /* Calculate boundary layer data... */
    read_met_pbl(met);

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

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

    /* Calculate convective available potential energy... */
    read_met_cape(clim, met);

    /* Detrending... */
    read_met_detrend(ctl, met);

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

  /* Read binary data... */
  else if (ctl->met_type >= 1 && ctl->met_type <= 4) {

    FILE *in;

    double r;

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

    /* Set timer... */
    SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);

    /* Open file... */
    if (!(in = fopen(filename, "r"))) {
      WARN("Cannot open file!");
      return 0;
    }

    /* Check type of binary data... */
    int met_type;
    FREAD(&met_type, int,
          1,
          in);
    if (met_type != ctl->met_type)
      ERRMSG("Wrong MET_TYPE of binary data!");

    /* Check version of binary data... */
    int version;
    FREAD(&version, int,
          1,
          in);
    if (version != 100)
      ERRMSG("Wrong version of binary data!");

    /* Read time... */
    FREAD(&met->time, double,
          1,
          in);
    jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
    LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
        met->time, year, mon, day, hour, min);
    if (year < 1900 || year > 2100 || mon < 1 || mon > 12
        || day < 1 || day > 31 || hour < 0 || hour > 23)
      ERRMSG("Error while reading time!");

    /* Read dimensions... */
    FREAD(&met->nx, int,
          1,
```

```
      in);
    LOG(2, "Number of longitudes: %d", met->nx);
    if (met->nx < 2 || met->nx > EX)
      ERRMSG("Number of longitudes out of range!");

    FREAD(&met->ny, int,
          1,
          in);
    LOG(2, "Number of latitudes: %d", met->ny);
    if (met->ny < 2 || met->ny > EY)
      ERRMSG("Number of latitudes out of range!");

    FREAD(&met->np, int,
          1,
          in);
    LOG(2, "Number of levels: %d", met->np);
    if (met->np < 2 || met->np > EP)
      ERRMSG("Number of levels out of range!");

    /* Read grid... */
    FREAD(met->lon, double,
            (size_t) met->nx,
          in);
    LOG(2, "Longitudes: %g, %g ... %g deg",
        met->lon[0], met->lon[1], met->lon[met->nx - 1]);

    FREAD(met->lat, double,
            (size_t) met->ny,
          in);
    LOG(2, "Latitudes: %g, %g ... %g deg",
        met->lat[0], met->lat[1], met->lat[met->ny - 1]);

    FREAD(met->p, double,
            (size_t) met->np,
          in);
    LOG(2, "Altitude levels: %g, %g ... %g km",
        Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
    LOG(2, "Pressure levels: %g, %g ... %g hPa",
        met->p[0], met->p[1], met->p[met->np - 1]);

    /* Read surface data... */
    read_met_bin_2d(in, met, met->ps, "PS");
    read_met_bin_2d(in, met, met->ts, "TS");
    read_met_bin_2d(in, met, met->zs, "ZS");
    read_met_bin_2d(in, met, met->us, "US");
    read_met_bin_2d(in, met, met->vs, "VS");
    read_met_bin_2d(in, met, met->pbl, "PBL");
    read_met_bin_2d(in, met, met->pt, "PT");
    read_met_bin_2d(in, met, met->tt, "TT");
    read_met_bin_2d(in, met, met->zt, "ZT");
    read_met_bin_2d(in, met, met->h2ot, "H2OT");
    read_met_bin_2d(in, met, met->pct, "PCT");
    read_met_bin_2d(in, met, met->pcb, "PCB");
    read_met_bin_2d(in, met, met->cl, "CL");
    read_met_bin_2d(in, met, met->plcl, "PLCL");
    read_met_bin_2d(in, met, met->plfc, "PLFC");
    read_met_bin_2d(in, met, met->pel, "PEL");
    read_met_bin_2d(in, met, met->cape, "CAPE");
    read_met_bin_2d(in, met, met->cin, "CIN");

    /* Read level data... */
    read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
    read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
    read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
    read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
    read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
    read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
    read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
    read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
    read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
    read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);

    /* Read final flag... */
    int final;
    FREAD(&final, int,
          1,
          in);
    if (final != 999)
      ERRMSG("Error while reading binary data!");

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

  /* Not implemented... */
  else
    ERRMSG("MET_TYPE not implemented!");
```

```
  /* Copy wind data to cache... */
#ifdef UVW
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++) {
        met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
        met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
        met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
      }
#endif

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

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

void read_met_bin_2d(
  FILE * in,
  met_t * met,
  float var[EX][EY],
  char *varname) {

  float *help;

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

  /* Read uncompressed... */
  LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
  FREAD(help, float,
            (size_t) (met->nx * met->ny),
        in);

  /* Copy data... */
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];

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

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

void read_met_bin_3d(
  FILE * in,
  ctl_t * ctl,
  met_t * met,
  float var[EX][EY][EP],
  char *varname,
  int precision,
  double tolerance) {

  float *help;

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

  /* Read uncompressed data... */
  if (ctl->met_type == 1) {
    LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
    FREAD(help, float,
              (size_t) (met->nx * met->ny * met->np),
          in);
  }

  /* Read packed data... */
  else if (ctl->met_type == 2)
    compress_pack(varname, help, (size_t) (met->ny * met->nx),
                  (size_t) met->np, 1, in);

  /* Read zfp data... */
  else if (ctl->met_type == 3) {
#ifdef ZFP
    compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
                 tolerance, 1, in);
#else
    ERRMSG("zfp compression not supported!");
    LOG(3, "%d %g", precision, tolerance);
#endif
  }

```

```
  /* Read zstd data... */
  else if (ctl->met_type == 4) {
#ifdef ZSTD
    compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
                  in);
#else
    ERRMSG("zstd compression not supported!");
#endif
  }

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];

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

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

void read_met_cape(
  clim_t * clim,
  met_t * met) {

  /* Set timer... */
  SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
  LOG(2, "Calculate CAPE...");

  /* Vertical spacing (about 100 m)... */
  const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
      int n = 0;
      double h2o = 0, t, theta = 0;
      double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
      double ptop = pbot - 50.;
      for (int ip = 0; ip < met->np; ip++) {
        if (met->p[ip] <= pbot) {
          theta += THETA(met->p[ip], met->t[ix][iy][ip]);
          h2o += met->h2o[ix][iy][ip];
          n++;
        }
        if (met->p[ip] < ptop && n > 0)
          break;
      }
      theta /= n;
      h2o /= n;

      /* Cannot compute anything if water vapor is missing... */
      met->plcl[ix][iy] = GSL_NAN;
      met->plfc[ix][iy] = GSL_NAN;
      met->pel[ix][iy] = GSL_NAN;
      met->cape[ix][iy] = GSL_NAN;
      met->cin[ix][iy] = GSL_NAN;
      if (h2o <= 0)
        continue;

      /* Find lifted condensation level (LCL)... */
      ptop = P(20.);
      pbot = met->ps[ix][iy];
      do {
        met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
        t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
        if (RH(met->plcl[ix][iy], t, h2o) > 100.)
          ptop = met->plcl[ix][iy];
        else
          pbot = met->plcl[ix][iy];
      } while (pbot - ptop > 0.1);

      /* Calculate CIN up to LCL... */
      INTPOL_INIT;
      double dcape, dz, h2o_env, t_env;
      double p = met->ps[ix][iy];
      met->cape[ix][iy] = met->cin[ix][iy] = 0;
      do {
        dz = dz0 * TVIRT(t, h2o);
        p /= pfac;
        t = theta / pow(1000. / p, 0.286);
        intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
```

```
                            &t_env, ci, cw, 1);
        intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
                            &h2o_env, ci, cw, 0);
        dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
          TVIRT(t_env, h2o_env) * dz;
        if (dcape < 0)
          met->cin[ix][iy] += fabsf((float) dcape);
      } while (p > met->plcl[ix][iy]);

      /* Calculate level of free convection (LFC), equilibrium level (EL),
         and convective available potential energy (CAPE)... */
      dcape = 0;
      p = met->plcl[ix][iy];
      t = theta / pow(1000. / p, 0.286);
      ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
      do {
        dz = dz0 * TVIRT(t, h2o);
        p /= pfac;
        t -= lapse_rate(t, h2o) * dz;
        double psat = PSAT(t);
        h2o = psat / (p - (1. - EPS) * psat);
        intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
                            &t_env, ci, cw, 1);
        intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
                            &h2o_env, ci, cw, 0);
        double dcape_old = dcape;
        dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
          TVIRT(t_env, h2o_env) * dz;
        if (dcape > 0) {
          met->cape[ix][iy] += (float) dcape;
          if (!isfinite(met->plfc[ix][iy]))
            met->plfc[ix][iy] = (float) p;
        } else if (dcape_old > 0)
          met->pel[ix][iy] = (float) p;
        if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
          met->cin[ix][iy] += fabsf((float) dcape);
      } while (p > ptop);

      /* Check results... */
      if (!isfinite(met->plfc[ix][iy]))
        met->cin[ix][iy] = GSL_NAN;
    }
}

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

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

  /* Set timer... */
  SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
  LOG(2, "Calculate cloud data...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

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

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

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

        /* Check ice water and liquid water content... */
        if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
            || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {

          /* Get cloud top pressure ... */
          met->pct[ix][iy]
            = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));

          /* Get cloud bottom pressure ... */
          if (!isfinite(met->pcb[ix][iy]))
            met->pcb[ix][iy]
              = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
        }

        /* 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_detrend(
  ctl_t * ctl,
  met_t * met) {

  met_t *help;

  /* Check parameters... */
  if (ctl->met_detrend <= 0)
    return;

  /* Set timer... */
  SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
  LOG(2, "Detrend meteo data...");

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

  /* Calculate standard deviation... */
  double sigma = ctl->met_detrend / 2.355;
  double tssq = 2. * SQR(sigma);

  /* Calculate box size in latitude... */
  int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
  sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);

  /* Calculate background... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++) {
    for (int iy = 0; iy < met->ny; iy++) {

      /* Calculate Cartesian coordinates... */
      double x0[3];
      geo2cart(0.0, met->lon[ix], met->lat[iy], x0);

      /* Calculate box size in longitude... */
      int sx =
        (int) (3. * DX2DEG(sigma, met->lat[iy]) /
               fabs(met->lon[1] - met->lon[0]));
      sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);

      /* Init... */
      float wsum = 0;
      for (int ip = 0; ip < met->np; ip++) {
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
      }

      /* Loop over neighboring grid points... */
      for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
        int ix3 = ix2;
        if (ix3 < 0)
          ix3 += met->nx;
        else if (ix3 >= met->nx)
          ix3 -= met->nx;
        for (int iy2 = GSL_MAX(iy - sy, 0);
             iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {

          /* Calculate Cartesian coordinates... */
          double x1[3];
          geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);

          /* Calculate weighting factor... */
          float w = (float) exp(-DIST2(x0, x1) / tssq);

          /* Add data... */
          wsum += w;
          for (int ip = 0; ip < met->np; ip++) {
            help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
            help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
            help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
            help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
          }
        }
      }

      /* Normalize... */
```

```
      for (int ip = 0; ip < met->np; ip++) {
        help->t[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Subtract background... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++) {
        met->t[ix][iy][ip] -= help->t[ix][iy][ip];
        met->u[ix][iy][ip] -= help->u[ix][iy][ip];
        met->v[ix][iy][ip] -= help->v[ix][iy][ip];
        met->w[ix][iy][ip] -= help->w[ix][iy][ip];
      }

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

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

void read_met_extrapolate(
  met_t * met) {

  /* Set timer... */
  SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
  LOG(2, "Extrapolate meteo data...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      int ip0;
      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 (int 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(
  ctl_t * ctl,
  met_t * met) {

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

  double logp[EP];

  int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;

  /* Set timer... */
  SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
  LOG(2, "Calculate geopotential heights...");

  /* Calculate log pressure... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < met->np; ip++)
    logp[ip] = log(met->p[ip]);

  /* Apply hydrostatic equation to calculate geopotential heights... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

```

```
      /* Get surface height and pressure... */
      double zs = met->zs[ix][iy];
      double lnps = log(met->ps[ix][iy]);

      /* Get temperature and water vapor vmr at the surface... */
      int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
      double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
                      met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
      double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
                        met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (zs +
                   ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
                         met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
      for (int ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] +
                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
                           met->h2o[ix][iy][ip - 1], logp[ip],
                           met->t[ix][iy][ip], met->h2o[ix][iy][ip]));

      /* Lower part of profile... */
      met->z[ix][iy][ip0]
        = (float) (zs +
                   ZDIFF(lnps, ts, h2os, logp[ip0],
                         met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
      for (int ip = ip0 - 1; ip >= 0; ip--)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip + 1] +
                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
                           met->h2o[ix][iy][ip + 1], logp[ip],
                           met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
    }

  /* Check control parameters... */
  if (dx == 0 || dy == 0)
    return;

  /* Default smoothing parameters... */
  if (dx < 0 || dy < 0) {
    if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
      dx = 3;
      dy = 2;
    } else {
      dx = 6;
      dy = 4;
    }
  }

  /* Calculate weights for smoothing... */
  float ws[dx + 1][dy + 1];
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix <= dx; ix++)
    for (int iy = 0; iy < dy; iy++)
      ws[ix][iy] = (1.0f - (float) ix / (float) dx)
        * (1.0f - (float) iy / (float) dy);

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        help[ip][ix][iy] = met->z[ix][iy][ip];

  /* Horizontal smoothing... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ip = 0; ip < met->np; ip++)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        float res = 0, wsum = 0;
        int iy0 = GSL_MAX(iy - dy + 1, 0);
        int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
        for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
          int ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (int iy2 = iy0; iy2 <= iy1; ++iy2)
            if (isfinite(help[ip][ix3][iy2])) {
              float w = ws[abs(ix - ix2)][abs(iy - iy2)];
              res += w * help[ip][ix3][iy2];
              wsum += w;
            }
        }
```

```
        if (wsum > 0)
          met->z[ix][iy][ip] = res / wsum;
        else
          met->z[ix][iy][ip] = GSL_NAN;
      }
}

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

void read_met_grid(
  char *filename,
  int ncid,
  ctl_t * ctl,
  met_t * met) {

  char levname[LEN], tstr[10];

  double rtime, r2;

  int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;

  size_t np;

  /* Set timer... */
  SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
  LOG(2, "Read meteo grid information...");

  /* MPTRAC meteo files... */
  if (ctl->clams_met_data == 0) {

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

    /* Check time information from data file... */
    if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
      NC(nc_get_var_double(ncid, varid, &rtime));
      if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
        WARN("Time information in meteo file does not match filename!");
    } else
      WARN("Time information in meteo file is missing!");
  }

  /* CLaMS meteo files... */
  else {

    /* Read time from file... */
    NC_GET_DOUBLE("time", &rtime, 0);

    /* Get time from filename (considering the century)... */
    if (rtime < 0)
      sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
    else
      sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
    hour = atoi(tstr);
    time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
  }

  /* Check time... */
  if (year < 1900 || year > 2100 || mon < 1 || mon > 12
      || day < 1 || day > 31 || hour < 0 || hour > 23)
    ERRMSG("Cannot read time from filename!");
  jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
  LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
      met->time, year2, mon2, day2, hour2, min2);

  /* Get grid dimensions... */
  NC_INQ_DIM("lon", &met->nx, 2, EX);
  LOG(2, "Number of longitudes: %d", met->nx);

  NC_INQ_DIM("lat", &met->ny, 2, EY);
  LOG(2, "Number of latitudes: %d", met->ny);

```

```
  if (ctl->vert_coord_met == 0) {
    int dimid;
    sprintf(levname, "lev");
    if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR)
      sprintf(levname, "plev");
  } else
    sprintf(levname, "hybrid");
  NC_INQ_DIM(levname, &met->np, 1, EP);
  if (met->np == 1) {
    int dimid;
    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));
    met->np = (int) np;
  }
  LOG(2, "Number of levels: %d", met->np);
  if (met->np < 2 || met->np > EP)
    ERRMSG("Number of levels out of range!");

  /* Read longitudes and latitudes... */
  NC_GET_DOUBLE("lon", met->lon, 1);
  LOG(2, "Longitudes: %g, %g ... %g deg",
      met->lon[0], met->lon[1], met->lon[met->nx - 1]);
  NC_GET_DOUBLE("lat", met->lat, 1);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      met->lat[0], met->lat[1], met->lat[met->ny - 1]);

  /* Read pressure levels... */
  if (ctl->met_np <= 0) {
    NC_GET_DOUBLE(levname, met->p, 1);
    for (int ip = 0; ip < met->np; ip++)
      met->p[ip] /= 100.;
    LOG(2, "Altitude levels: %g, %g ... %g km",
        Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
    LOG(2, "Pressure levels: %g, %g ... %g hPa",
        met->p[0], met->p[1], met->p[met->np - 1]);
  }
}

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

void read_met_levels(
  int ncid,
  ctl_t * ctl,
  met_t * met) {

  /* Set timer... */
  SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
  LOG(2, "Read level data...");

  /* MPTRAC meteo data... */
  if (ctl->clams_met_data == 0) {

    /* Read meteo data... */
    if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
      ERRMSG("Cannot read temperature!");
    if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
      ERRMSG("Cannot read zonal wind!");
    if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
      ERRMSG("Cannot read meridional wind!");
    if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
      WARN("Cannot read vertical velocity!");
    if (!read_met_nc_3d
        (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
      WARN("Cannot read specific humidity!");
    if (!read_met_nc_3d
        (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
      WARN("Cannot read ozone data!");
    if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
        WARN("Cannot read cloud liquid water content!");
      if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
        WARN("Cannot read cloud ice water content!");
    }
    if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d
          (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud rain water content!");
      if (!read_met_nc_3d
          (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud snow water content!");
    }
```

```
    if (ctl->met_relhum) {
      if (!read_met_nc_3d(ncid, "rh", "RH", ctl, met, met->h2o, 0.01f, 1))
        WARN("Cannot read relative humidity!");
#pragma omp parallel for default(shared) collapse(2)
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          for (int ip = 0; ip < met->np; ip++) {
            double pw = met->h2o[ix][iy][ip] * PSAT(met->t[ix][iy][ip]);
            met->h2o[ix][iy][ip] =
              (float) (pw / (met->p[ip] - (1.0 - EPS) * pw));
          }
    }

    /* Transfer from model levels to pressure levels... */
    if (ctl->met_np > 0) {

      /* Read pressure on model levels... */
      if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->pl, 0.01f, 1))
        ERRMSG("Cannot read pressure on model levels!");

      /* Vertical interpolation from model 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 new pressure levels... */
      met->np = ctl->met_np;
      for (int ip = 0; ip < met->np; ip++)
        met->p[ip] = ctl->met_p[ip];
    }

  }

  /* CLaMS meteo data... */
  else if (ctl->clams_met_data == 1) {

    /* Read meteorological data... */
    if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
      ERRMSG("Cannot read temperature!");
    if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
      ERRMSG("Cannot read zonal wind!");
    if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
      ERRMSG("Cannot read meridional wind!");
    if (!read_met_nc_3d(ncid, "W", "OMEGA", ctl, met, met->w, 0.01f, 1))
      WARN("Cannot read vertical velocity!");
    if (!read_met_nc_3d(ncid, "ZETA", "zeta", ctl, met, met->zeta, 1.0, 1))
      WARN("Cannot read ZETA in meteo data!");
    if (ctl->vert_vel == 1) {
      if (!read_met_nc_3d
          (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
           0.00001157407f, 1)) {
        if (!read_met_nc_3d
            (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
             0.00001157407f, 1)) {
          WARN("Cannot read vertical velocity!");
        }
      }
    }
    if (!read_met_nc_3d
        (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
      WARN("Cannot read specific humidity!");
    if (!read_met_nc_3d
        (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
      WARN("Cannot read ozone data!");
    if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
        WARN("Cannot read cloud liquid water content!");
      if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
        WARN("Cannot read cloud ice water content!");
    }
    if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d
          (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud rain water content!");
      if (!read_met_nc_3d
          (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud snow water content!");
    }

    /* Transfer from model levels to pressure levels... */
```

```
  if (ctl->met_np > 0) {

    /* Read pressure on model levels... */
    if (!read_met_nc_3d(ncid, "pl", "PRESS", ctl, met, met->pl, 1.0, 1))
      ERRMSG("Cannot read pressure on model levels!");

    /* Vertical interpolation from model 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);
    if (ctl->vert_vel == 1) {
      read_met_ml2pl(ctl, met, met->zeta);
      read_met_ml2pl(ctl, met, met->zeta_dot);
    }

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

    /* Create a pressure field... */
    for (int i = 0; i < met->nx; i++)
      for (int j = 0; j < met->ny; j++)
        for (int k = 0; k < met->np; k++) {
          met->patp[i][j][k] = (float) met->p[k];
        }
  }
} else
  ERRMSG("Meteo data format unknown!");

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

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

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

  double aux[EP], p[EP];

  /* Set timer... */
  SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
  LOG(2, "Interpolate meteo data to pressure levels...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(aux,p) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

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

      /* Interpolate... */
      for (int ip = 0; ip < ctl->met_np; ip++) {
        double 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];
        int 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 (int ip = 0; ip < ctl->met_np; ip++)
        var[ix][iy][ip] = (float) aux[ip];
    }
}

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

int read_met_nc_2d(
  int ncid,
  char *varname,
```

```
  char *varname2,
  ctl_t * ctl,
  met_t * met,
  float dest[EX][EY],
  float scl,
  int init) {

  char varsel[LEN];

  float offset, scalfac;

  int varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
      WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
      return 0;
    } else {
      sprintf(varsel, "%s", varname2);
  } else
    sprintf(varsel, "%s", varname);

  /* Read packed data... */
  if (ctl->met_nc_scale
      && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
      && nc_get_att_float(ncid, varid, "scale_factor",
                          &scalfac) == NC_NOERR) {

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

    /* Read fill value and missing value... */
    short fillval, missval;
    if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 2-D variable: %s"
        " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
        varsel, fillval, missval, scalfac, offset);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        if (init)
          dest[ix][iy] = 0;
        short aux = help[ARRAY_2D(iy, ix, met->nx)];
        if ((fillval == 0 || aux != fillval)
            && (missval == 0 || aux != missval)
            && fabsf(aux * scalfac + offset) < 1e14f)
          dest[ix][iy] += scl * (aux * scalfac + offset);
        else
          dest[ix][iy] = GSL_NAN;
      }

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

  /* Unpacked data... */
  else {

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

    /* Read fill value and missing value... */
    float fillval, missval;
    if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
        varsel, fillval, missval);

```

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        if (init)
          dest[ix][iy] = 0;
        float aux = help[ARRAY_2D(iy, ix, met->nx)];
        if ((fillval == 0 || aux != fillval)
            && (missval == 0 || aux != missval)
            && fabsf(aux) < 1e14f)
          dest[ix][iy] += scl * aux;
        else
          dest[ix][iy] = GSL_NAN;
      }

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

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

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

int read_met_nc_3d(
  int ncid,
  char *varname,
  char *varname2,
  ctl_t * ctl,
  met_t * met,
  float dest[EX][EY][EP],
  float scl,
  int init) {

  char varsel[LEN];

  float offset, scalfac;

  int varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
      WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
      return 0;
    } else {
      sprintf(varsel, "%s", varname2);
  } else
    sprintf(varsel, "%s", varname);

  /* Read packed data... */
  if (ctl->met_nc_scale
      && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
      && nc_get_att_float(ncid, varid, "scale_factor",
                          &scalfac) == NC_NOERR) {

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

    /* Read fill value and missing value... */
    short fillval, missval;
    if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 3-D variable: %s "
        "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
        varsel, fillval, missval, scalfac, offset);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        for (int ip = 0; ip < met->np; ip++) {
          if (init)
            dest[ix][iy][ip] = 0;
```

```
          short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
          if ((fillval == 0 || aux != fillval)
              && (missval == 0 || aux != missval)
              && fabsf(aux * scalfac + offset) < 1e14f)
            dest[ix][iy][ip] += scl * (aux * scalfac + offset);
          else
            dest[ix][iy][ip] = GSL_NAN;
        }

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

  /* Unpacked data... */
  else {

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

    /* Read fill value and missing value... */
    float fillval, missval;
    if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
        varsel, fillval, missval);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        for (int ip = 0; ip < met->np; ip++) {
          if (init)
            dest[ix][iy][ip] = 0;
          float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
          if ((fillval == 0 || aux != fillval)
              && (missval == 0 || aux != missval)
              && fabsf(aux) < 1e14f)
            dest[ix][iy][ip] += scl * aux;
          else
            dest[ix][iy][ip] = GSL_NAN;
        }

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

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

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

void read_met_pbl(
  met_t * met) {

  /* Set timer... */
  SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
  LOG(2, "Calculate planetary boundary layer...");

  /* Parameters used to estimate the height of the PBL
     (e.g., Vogelezang and Holtslag, 1996; Seidel et al., 2012)... */
  const double rib_crit = 0.25, dz = 0.05, umin = 5.0;

  /* Loop over grid points... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Set bottom level of PBL... */
      double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);

      /* Find lowest level near the bottom... */
      int ip;
      for (ip = 1; ip < met->np; ip++)
        if (met->p[ip] < pbl_bot)
          break;

      /* Get near surface data... */
```

```
      double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
                      met->p[ip], met->z[ix][iy][ip], pbl_bot);
      double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
                      met->p[ip], met->t[ix][iy][ip], pbl_bot);
      double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
                      met->p[ip], met->u[ix][iy][ip], pbl_bot);
      double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
                      met->p[ip], met->v[ix][iy][ip], pbl_bot);
      double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
                        met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
      double tvs = THETAVIRT(pbl_bot, ts, h2os);

      /* Init... */
      double rib_old = 0;

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

        /* Get squared horizontal wind speed... */
        double vh2
          = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
        vh2 = GSL_MAX(vh2, SQR(umin));

        /* Calculate bulk Richardson number... */
        double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
          * (THETAVIRT(met->p[ip], met->t[ix][iy][ip],
                       met->h2o[ix][iy][ip]) - tvs) / vh2;

        /* Check for critical value... */
        if (rib >= rib_crit) {
          met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
                                          rib, met->p[ip], rib_crit));
          if (met->pbl[ix][iy] > pbl_bot)
            met->pbl[ix][iy] = (float) pbl_bot;
          break;
        }

        /* Save Richardson number... */
        rib_old = rib;
      }
    }
}

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

void read_met_periodic(
  met_t * met) {

  /* Set timer... */
  SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
  LOG(2, "Apply periodic boundary conditions...");

  /* 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];
    met->ts[met->nx - 1][iy] = met->ts[0][iy];
    met->us[met->nx - 1][iy] = met->us[0][iy];
    met->vs[met->nx - 1][iy] = met->vs[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 pows[EP];

  /* Set timer... */
  SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
  LOG(2, "Calculate potential vorticity...");

  /* Set powers... */
#pragma omp parallel for default(shared)
  for (int 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)
  for (int ix = 0; ix < met->nx; ix++) {

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

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

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

      /* Set auxiliary variables... */
      double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
      double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
      double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
      double c0 = cos(met->lat[iy0] / 180. * M_PI);
      double c1 = cos(met->lat[iy1] / 180. * M_PI);
      double cr = cos(latr / 180. * M_PI);
      double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);

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

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

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

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

        /* Get gradients in pressure... */
        double dtdp, dudp, dvdp;
        double dp0 = 100. * (met->p[ip] - met->p[ip0]);
        double dp1 = 100. * (met->p[ip1] - met->p[ip]);
        if (ip != ip0 && ip != ip1) {
          double 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 {
          double 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)
  for (int ix = 0; ix < met->nx; ix++)
    for (int 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;

  /* 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;

  /* Set timer... */
  SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
  LOG(2, "Downsampling of meteo data...");

  /* 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 (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
    for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
      for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
        help->ps[ix][iy] = 0;
        help->zs[ix][iy] = 0;
        help->ts[ix][iy] = 0;
        help->us[ix][iy] = 0;
        help->vs[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;
        float wsum = 0;
        for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
             ix2++) {
          int ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
            for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
              float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
                * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
                * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->zs[ix][iy] += w * met->zs[ix3][iy2];
              help->ts[ix][iy] += w * met->ts[ix3][iy2];
              help->us[ix][iy] += w * met->us[ix3][iy2];
              help->vs[ix][iy] += w * met->vs[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->ts[ix][iy] /= wsum;
        help->us[ix][iy] /= wsum;
        help->vs[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 (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (int 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->ts[met->nx][met->ny] = help->ts[ix][iy];
      met->us[met->nx][met->ny] = help->us[ix][iy];
      met->vs[met->nx][met->ny] = help->vs[ix][iy];
      met->np = 0;
      for (int 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,
  ctl_t * ctl) {

  /* Set timer... */
  SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
  LOG(2, "Read surface data...");

  /* MPTRAC meteo data... */
  if (ctl->clams_met_data == 0) {

    /* Read surface pressure... */
    if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
      if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
        if (!read_met_nc_2d(ncid, "sp", "SP", ctl, met, met->ps, 0.01f, 1)) {
          WARN("Cannot not read surface pressure data (use lowest level)!");
          for (int ix = 0; ix < met->nx; ix++)
            for (int iy = 0; iy < met->ny; iy++)
              met->ps[ix][iy] = (float) met->p[0];
        }
      }
    } else
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);

```

```
    /* Read geopotential height at the surface... */
    if (!read_met_nc_2d
        (ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
      if (!read_met_nc_2d
          (ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
        WARN("Cannot read surface geopotential height!");

    /* Read temperature at the surface... */
    if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
      WARN("Cannot read surface temperature!");

    /* Read zonal wind at the surface... */
    if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
      WARN("Cannot read surface zonal wind!");

    /* Read meridional wind at the surface... */
    if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
      WARN("Cannot read surface meridional wind!");
  }

  /* CLaMS meteo data... */
  else {

    /* Read surface pressure... */
    if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
      WARN("Cannot not read surface pressure data (use lowest level)!");
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) met->p[0];
    }

    /* Read geopotential height at the surface
       (use lowermost level of 3-D data field)... */
    float *help;
    ALLOC(help, float,
          EX * EY * EP);
    memcpy(help, met->pl, sizeof(met->pl));
    if (!read_met_nc_3d
        (ncid, "gph", "GPH", ctl, met, met->pl, (float) (1e-3 / G0), 1)) {
      ERRMSG("Cannot read geopotential height!");
    } else
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->zs[ix][iy] = met->pl[ix][iy][0];
    memcpy(met->pl, help, sizeof(met->pl));
    free(help);

    /* Read temperature at the surface... */
    if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
      WARN("Cannot read surface temperature!");

    /* Read zonal wind at the surface... */
    if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
      WARN("Cannot read surface zonal wind!");

    /* Read meridional wind at the surface... */
    if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
      WARN("Cannot read surface meridional wind!");
  }
}

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

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

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

  /* Set timer... */
  SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
  LOG(2, "Calculate tropopause...");

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

```
#pragma omp parallel for default(shared) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int 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) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = (float) clim_tropo(clim, 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(t,t2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        for (int iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);

        /* Find minimum... */
        int 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(t,t2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        int iz;
        for (iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);

        /* Find 1st tropopause... */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++) {
          int found = 1;
          for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
            if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
                ctl->met_tropo_lapse) {
              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++) {
            int found = 1;
            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
                  ctl->met_tropo_lapse_sep) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 170; iz++) {
            int found = 1;
            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
                  ctl->met_tropo_lapse) {
                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(pv,pv2,th,th2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate potential vorticity profile... */
        for (int iz = 0; iz < met->np; iz++)
          pv[iz] = met->pv[ix][iy][iz];
        spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);

        /* Interpolate potential temperature profile... */
        for (int 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, ctl->met_tropo_spline);

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

  else
    ERRMSG("Cannot calculate tropopause!");

  /* Interpolate temperature, geopotential height, and water vapor vmr... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {
      double h2ot, tt, zt;
      INTPOL_INIT;
      intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &tt, ci, cw, 1);
      intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &zt, ci, cw, 0);
      intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &h2ot, ci, cw, 0);
      met->tt[ix][iy] = (float) tt;
      met->zt[ix][iy] = (float) zt;
      met->h2ot[ix][iy] = (float) h2ot;
    }
}

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

void read_obs(
  char *filename,
  double *rt,
  double *rz,
  double *rlon,
  double *rlat,
  double *robs,
  int *nobs) {

  FILE *in;

  char line[LEN];

  /* Open observation data file... */
  LOG(1, "Read observation data: %s", filename);
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read observations... */
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
               &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
      if ((++(*nobs)) >= NOBS)
```

```
      ERRMSG("Too many observations!");

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

  /* Check time... */
  for (int i = 1; i < *nobs; i++)
    if (rt[i] < rt[i - 1])
      ERRMSG("Time must be ascending!");

  /* Write info... */
  int n = *nobs;
  double mini, maxi;
  LOG(2, "Number of observations: %d", *nobs);
  gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
  LOG(2, "Observation range: %g ... %g", mini, maxi);
}

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

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

  FILE *in = NULL;

  char fullname1[LEN], fullname2[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) {
    char dummy[LEN], line[LEN], rvarname[LEN];
    while (fgets(line, LEN, in)) {
      if (sscanf(line, "%4999s %4999s %4999s", 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
      ERRMSG("Missing variable %s!\n", fullname1);
```

```
  }

  /* Write info... */
  LOG(1, "%s = %s", fullname1, rval);

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

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

double sedi(
  double p,
  double T,
  double rp,
  double rhop) {

  /* Convert particle radius from microns to m... */
  rp *= 1e-6;

  /* Density of dry air [kg / m^3]... */
  double rho = RHO(p, T);

  /* Dynamic viscosity of air [kg / (m s)]... */
  double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Thermal velocity of an air molecule [m / s]... */
  double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));

  /* Mean free path of an air molecule [m]... */
  double lambda = 2. * eta / (rho * v);

  /* Knudsen number for air (dimensionless)... */
  double K = lambda / rp;

  /* Cunningham slip-flow correction (dimensionless)... */
  double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));

  /* Sedimentation velocity [m / s]... */
  return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
}

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

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

  /* Cubic spline interpolation... */
  if (method == 1) {

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

    /* Interpolate 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);
  }

  /* Linear interpolation... */
  else {
    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 {
```

```
        int idx = locate_irr(x, n, x2[i]);
        y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
      }
  }
}

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

float stddev(
  float *data,
  int n) {

  if (n <= 0)
    return 0;

  float mean = 0, var = 0;

  for (int i = 0; i < n; ++i) {
    mean += data[i];
    var += SQR(data[i]);
  }

  var = var / (float) n - SQR(mean / (float) n);

  return (var > 0 ? sqrtf(var) : 0);
}

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

double sza(
  double sec,
  double lon,
  double lat) {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [rad]... */
  return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
}

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

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,
  const char *group,
  int output) {

  static char names[NTIMER][100], groups[NTIMER][100];

  static double rt_name[NTIMER], rt_group[NTIMER],
    rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;

  static int iname = -1, igroup = -1, nname, ngroup, ct_name[NTIMER];

  /* Get time... */
  t1 = omp_get_wtime();
  dt = t1 - t0;

  /* Add elapsed time to current timers... */
  if (iname >= 0) {
    rt_name[iname] += dt;
    rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
    rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
    ct_name[iname]++;
  }
  if (igroup >= 0)
    rt_group[igroup] += t1 - t0;

  /* Report timers... */
  if (output) {
    for (int i = 0; i < nname; i++)
      LOG(1, "TIMER_%s = %.3f s    (min= %g s, mean= %g s,"
          " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
          rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
    for (int i = 0; i < ngroup; i++)
      LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
    double total = 0.0;
    for (int i = 0; i < nname; i++)
      total += rt_name[i];
    LOG(1, "TIMER_TOTAL = %.3f s", total);
  }

  /* Identify IDs of next timer... */
  for (iname = 0; iname < nname; iname++)
    if (strcasecmp(name, names[iname]) == 0)
      break;
  for (igroup = 0; igroup < ngroup; igroup++)
    if (strcasecmp(group, groups[igroup]) == 0)
      break;

  /* Check whether this is a new timer... */
  if (iname >= nname) {
    sprintf(names[iname], "%s", name);
    if ((++nname) > NTIMER)
      ERRMSG("Too many timers!");
  }

  /* Check whether this is a new group... */
  if (igroup >= ngroup) {
    sprintf(groups[igroup], "%s", group);
    if ((++ngroup) > NTIMER)
      ERRMSG("Too many groups!");
  }

  /* Save starting time... */
  t0 = t1;
}

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

double tropo_weight(
  clim_t * clim,
  double t,
  double lat,
```

```
  double p) {

  /* Get tropopause pressure... */
  double pt = clim_tropo(clim, t, lat);

  /* Get pressure range... */
  double p1 = pt * 0.866877899;
  double p0 = pt / 0.866877899;

  /* Get weighting factor... */
  if (p > p0)
    return 1;
  else if (p < p1)
    return 0;
  else
    return LIN(p0, 1.0, p1, 0.0, p);
}

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

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

  /* Set timer... */
  SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write atmospheric data: %s", filename);

  /* Write ASCII data... */
  if (ctl->atm_type == 0)
    write_atm_asc(filename, ctl, atm, t);

  /* Write binary data... */
  else if (ctl->atm_type == 1)
    write_atm_bin(filename, ctl, atm);

  /* Write netCDF data... */
  else if (ctl->atm_type == 2)
    write_atm_nc(filename, ctl, atm);

  /* Write CLaMS data... */
  else if (ctl->atm_type == 3)
    write_atm_clams(ctl, atm, t);

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of particles: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  for (int iq = 0; iq < ctl->nq; iq++) {
    char msg[LEN];
    sprintf(msg, "Quantity %s range: %s ... %s %s",
            ctl->qnt_name[iq], ctl->qnt_format[iq],
            ctl->qnt_format[iq], ctl->qnt_unit[iq]);
    gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
    LOG(2, msg, mini, maxi);
  }
}

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

void write_atm_asc(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm,
  double t) {

  FILE *out;

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

```
  /* 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... */
    double r;
    int year, mon, day, hour, min, sec;
    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... */
    FILE *in;
    if (!(in = fopen(ctl->atm_gpfile, "r")))
      ERRMSG("Cannot open file!");
    char line[LEN];
    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 (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");

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

    /* Check time... */
    if (ctl->atm_filter == 2 && (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 (int iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
        fprintf(out, ctl->qnt_format[iq], GSL_NAN);
      else
        fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

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

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

void write_atm_bin(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm) {

  FILE *out;

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

  /* Write version of binary data... */
  int version = 100;
  FWRITE(&version, int,
         1,
         out);
```

```

  /* 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 (int iq = 0; iq < ctl->nq; iq++)
    FWRITE(atm->q[iq], double,
              (size_t) atm->np,
           out);

  /* Write final flag... */
  int final = 999;
  FWRITE(&final, int,
         1,
         out);

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

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

void write_atm_clams(
  ctl_t * ctl,
  atm_t * atm,
  double t) {

  /* Global Counter... */
  static size_t out_cnt = 0;

  char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";

  double r, r_start, r_stop;

  int year, mon, day, hour, min, sec;
  int year_start, mon_start, day_start, hour_start, min_start, sec_start;
  int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
  int ncid, varid, tid, pid, cid, zid, dim_ids[2];

  /* time, nparc */
  size_t start[2], count[2];

  /* Determine start and stop times of calculation... */
  jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
  jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
            &min_start, &sec_start, &r_start);
  jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
            &min_stop, &sec_stop, &r_stop);

  /* Set filename... */
  sprintf(filename_out,
          "./traj_fix_3d_%02d%02d%02d%02d_%02d%02d%02d%02d.nc",
          year_start % 100, mon_start, day_start, hour_start,
          year_stop % 100, mon_stop, day_stop, hour_stop);
  printf("Write traj file: %s\n", filename_out);

  /* Define hyperslap for the traj_file... */
  start[0] = out_cnt;
  start[1] = 0;
  count[0] = 1;
  count[1] = (size_t) atm->np;

  /* Create the file at the first timestep... */
  if (out_cnt == 0) {

    /* Create file... */
    nc_create(filename_out, NC_CLOBBER, &ncid);

    /* Define dimensions... */
    NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
    NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
    NC(nc_def_dim(ncid, "TMDT", 7, &cid));
    dim_ids[0] = tid;
    dim_ids[1] = pid;

```

```
    /* Define variables and their attributes... */
    NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
               "seconds since 2000-01-01 00:00:00 UTC");
    NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
    NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
    NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
    NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);

    /* Define global attributes... */
    NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
    NC_PUT_ATT_GLOBAL("model", "MPTRAC");

    /* End definitions... */
    NC(nc_enddef(ncid));
    NC(nc_close(ncid));
  }

  /* Increment global counter to change hyperslap... */
  out_cnt++;

  /* Open file... */
  NC(nc_open(filename_out, NC_WRITE, &ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", atm->time, 1);
  NC_PUT_DOUBLE("LAT", atm->lat, 1);
  NC_PUT_DOUBLE("LON", atm->lon, 1);
  NC_PUT_DOUBLE("PRESS", atm->p, 1);
  if (ctl->vert_coord_ap == 1) {
    NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
  } else if (ctl->qnt_zeta >= 0) {
    NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
  }
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);

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

  /* At the last time step create the init_fix_YYYYMMDDHH file... */
  if ((year == year_stop) && (mon == mon_stop)
      && (day == day_stop) && (hour == hour_stop)) {

    /* Set filename... */
    char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
    sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
            year_stop % 100, mon_stop, day_stop, hour_stop);
    printf("Write init file: %s\n", filename_init);

    /* Create file... */
    nc_create(filename_init, NC_CLOBBER, &ncid);

    /* Define dimensions... */
    NC(nc_def_dim(ncid, "time", 1, &tid));
    NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
    dim_ids[0] = tid;
    dim_ids[1] = pid;

    /* Define variables and their attributes... */
    NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
               "seconds since 2000-01-01 00:00:00 UTC");
    NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
    NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
    NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
    NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
    NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
    NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);

    /* Define global attributes... */
    NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
    NC_PUT_ATT_GLOBAL("model", "MPTRAC");

    /* End definitions... */
    NC(nc_enddef(ncid));

    /* Write data... */
    NC_PUT_DOUBLE("time", atm->time, 0);
    NC_PUT_DOUBLE("LAT", atm->lat, 0);
    NC_PUT_DOUBLE("LON", atm->lon, 0);
    NC_PUT_DOUBLE("PRESS", atm->p, 0);
    NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
```

```
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

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

void write_atm_nc(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm) {

  int ncid, obsid, varid;

  size_t start[2], count[2];

  /* Create file... */
  nc_create(filename, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
               ctl->qnt_longname[iq], ctl->qnt_unit[iq]);

  /* Define global attributes... */
  NC_PUT_ATT_GLOBAL("featureType", "point");

  /* End definitions... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", atm->time, 0);
  NC_PUT_DOUBLE("press", atm->p, 0);
  NC_PUT_DOUBLE("lon", atm->lon, 0);
  NC_PUT_DOUBLE("lat", atm->lat, 0);
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

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

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

  static FILE *out;

  static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
    dlon, dlat, dz, x[NCSI], y[NCSI];

  static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;

  /* Set timer... */
  SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);

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

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

    /* Allocate... */
    ALLOC(area, double,
          ctl->csi_ny);
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
```

```
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create new file... */
    LOG(1, "Write CSI data: %s", 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 (ratio of forecasts and observations) [%%]\n"
            "# $8 = probability of detection (POD) [%%]\n"
            "# $9 = false alarm rate (FAR) [%%]\n"
            "# $10 = critical success index (CSI) [%%]\n");
    fprintf(out,
            "# $11 = hits associated with random chance\n"
            "# $12 = equitable threat score (ETS) [%%]\n"
            "# $13 = Pearson linear correlation coefficient\n"
            "# $14 = Spearman rank-order correlation coefficient\n"
            "# $15 = column density mean error (F - O) [kg/m^2]\n"
            "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
            "# $17 = column density mean absolute error [kg/m^2]\n"
            "# $18 = number of data points\n\n");

    /* Set grid box size... */
    dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
    dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
    dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;

    /* Set horizontal coordinates... */
    for (iy = 0; iy < ctl->csi_ny; iy++) {
      double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
      area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
    }
  }

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

  /* Allocate... */
  ALLOC(modmean, double,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
  ALLOC(obsmean, double,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
  ALLOC(obscount, int,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Check observation data... */
    if (!isfinite(robs[i]))
      continue;

    /* Calculate indices... */
    ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
    iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
    iz = (int) ((rz[i] - ctl->csi_z0) / dz);

    /* 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... */
    int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
    obsmean[idx] += robs[i];
    obscount[idx]++;
  }
```

```
  /* Analyze model data... */
  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) / dlon);
    iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
    iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);

    /* 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... */
    int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
    modmean[idx] += atm->q[ctl->qnt_m][ip];
  }

  /* Analyze all grid cells... */
  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... */
        int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
        if (obscount[idx] > 0)
          obsmean[idx] /= obscount[idx];

        /* Calculate column density... */
        if (modmean[idx] > 0)
          modmean[idx] /= (1e6 * area[iy]);

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

        /* Save data for other verification statistics... */
        if (obscount[idx] > 0
            && (obsmean[idx] >= ctl->csi_obsmin
                || modmean[idx] >= ctl->csi_modmin)) {
          x[n] = modmean[idx];
          y[n] = obsmean[idx];
          if ((++n) > NCSI)
            ERRMSG("Too many data points to calculate statistics!");
        }
      }

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

    /* Calculate verification statistics
       (https://www.cawcr.gov.au/projects/verification/) ... */
    static double work[2 * NCSI];
    int n_obs = cx + cy;
    int n_for = cx + cz;
    double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
    double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
    double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
    double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
    double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
    double ets = (cx + cy + cz - cx_rd > 0) ?
      (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
    double rho_p =
      (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
    double rho_s =
      (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
    for (int i = 0; i < n; i++)
      work[i] = x[i] - y[i];
    double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
    double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
                                                         0.0) : GSL_NAN;
    double absdev =
```

```
      (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;

    /* Write... */
    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %g %g %d\n",
            t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
            rho_p, rho_s, mean, rmse, absdev, n);

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

  /* Free... */
  free(modmean);
  free(obsmean);
  free(obscount);

  /* Finalize... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(area);
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}

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

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

  static FILE *out;

  static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
    x[3], zm[NENS];

  static int n[NENS];

  /* Set timer... */
  SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);

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

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

  /* Init... */
  for (int i = 0; i < NENS; i++) {
    for (int iq = 0; iq < ctl->nq; iq++)
      qm[iq][i] = qs[iq][i] = 0;
    xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
    n[i] = 0;
  }

  /* 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 ensemble ID... */
    if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
      ERRMSG("Ensemble ID is out of range!");

    /* Get means... */
    geo2cart(0, atm->lon[ip], atm->lat[ip], x);
    for (int iq = 0; iq < ctl->nq; iq++) {
      qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
      qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
    }
    xm[ctl->qnt_ens][0] += x[0];
    xm[ctl->qnt_ens][1] += x[1];
    xm[ctl->qnt_ens][2] += x[2];
    zm[ctl->qnt_ens] += Z(atm->p[ip]);
```

```
    n[ctl->qnt_ens]++;
  }

  /* Create file... */
  LOG(1, "Write ensemble data: %s", 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 (int iq = 0; iq < ctl->nq; iq++)
    fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
            ctl->qnt_name[iq], ctl->qnt_unit[iq]);
  for (int 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);

  /* Write data... */
  for (int i = 0; i < NENS; i++)
    if (n[i] > 0) {
      cart2geo(xm[i], &dummy, &lon, &lat);
      fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
      for (int iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
      }
      for (int iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
        fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
      }
      fprintf(out, " %d\n", n[i]);
    }

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

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

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

  double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;

  int *ixs, *iys, *izs, *np;

  /* Set timer... */
  SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write grid data: %s", filename);

  /* Allocate... */
  ALLOC(cd, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(mass, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(vmr_expl, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(vmr_impl, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(z, double,
        ctl->grid_nz);
  ALLOC(lon, double,
        ctl->grid_nx);
  ALLOC(lat, double,
        ctl->grid_ny);
  ALLOC(area, double,
        ctl->grid_ny);
  ALLOC(press, double,
        ctl->grid_nz);
  ALLOC(np, int,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(ixs, int,
        atm->np);
  ALLOC(iys, int,
        atm->np);
```

```
  ALLOC(izs, int,
        atm->np);

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

  /* Set vertical coordinates... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < ctl->grid_nz; iz++) {
    z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
    press[iz] = P(z[iz]);
  }

  /* Set horizontal coordinates... */
  for (int ix = 0; ix < ctl->grid_nx; ix++)
    lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
#pragma omp parallel for default(shared)
  for (int iy = 0; iy < ctl->grid_ny; iy++) {
    lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
    area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
      * cos(lat[iy] * M_PI / 180.);
  }

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

  /* Get grid box indices... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++) {
    ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
    iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
    izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
    if (atm->time[ip] < t0 || atm->time[ip] > t1
        || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
        || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
        || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
      izs[ip] = -1;
  }

  /* Average data... */
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0) {
      int idx =
        ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
      np[idx]++;
      if (ctl->qnt_m >= 0)
        mass[idx] += atm->q[ctl->qnt_m][ip];
      if (ctl->qnt_vmr >= 0)
        vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
    }

  /* Calculate column density and vmr... */
#pragma omp parallel for default(shared)
  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++) {

        /* Get grid index... */
        int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);

        /* Calculate column density... */
        cd[idx] = GSL_NAN;
        if (ctl->qnt_m >= 0)
          cd[idx] = mass[idx] / (1e6 * area[iy]);

        /* Calculate volume mixing ratio (implicit)... */
        vmr_impl[idx] = GSL_NAN;
        if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
          vmr_impl[idx] = 0;
          if (mass[idx] > 0) {

            /* Get temperature... */
            double temp;
            INTPOL_INIT;
            intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
                               lon[ix], lat[iy], &temp, ci, cw, 1);

            /* Calculate volume mixing ratio... */
            vmr_impl[idx] = MA / ctl->molmass * mass[idx]
              / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
          }
        }

        /* Calculate volume mixing ratio (explicit)... */
```

```
        if (ctl->qnt_vmr >= 0 && np[idx] > 0)
          vmr_expl[idx] /= np[idx];
        else
          vmr_expl[idx] = GSL_NAN;
      }

  /* Write ASCII data... */
  if (ctl->grid_type == 0)
    write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
                   t, z, lon, lat, area, dz, np);

  /* Write netCDF data... */
  else if (ctl->grid_type == 1)
    write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
                  t, z, lon, lat, area, dz, np);

  /* Error message... */
  else
    ERRMSG("Grid data format GRID_TYPE unknown!");

  /* Free... */
  free(cd);
  free(mass);
  free(vmr_expl);
  free(vmr_impl);
  free(z);
  free(lon);
  free(lat);
  free(area);
  free(press);
  free(np);
  free(ixs);
  free(iys);
  free(izs);
}

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

void write_grid_asc(
  const char *filename,
  ctl_t * ctl,
  double *cd,
  double *vmr_expl,
  double *vmr_impl,
  double t,
  double *z,
  double *lon,
  double *lat,
  double *area,
  double dz,
  int *np) {

  FILE *in, *out;

  char line[LEN];

  /* Check if gnuplot output is requested... */
  if (ctl->grid_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... */
    double r;
    int year, mon, day, hour, min, sec;
    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 {

    /* 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 depth [km]\n"
          "# $7 = number of particles [1]\n"
          "# $8 = column density (implicit) [kg/m^2]\n"
          "# $9 = volume mixing ratio (implicit) [ppv]\n"
          "# $10 = volume mixing ratio (explicit) [ppv]\n\n");

  /* Write data... */
  for (int ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
      fprintf(out, "\n");
    for (int iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
        fprintf(out, "\n");
      for (int iz = 0; iz < ctl->grid_nz; iz++) {
        int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
        if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
          fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
                  t, z[iz], lon[ix], lat[iy], area[iy], dz,
                  np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
      }
    }
  }

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

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

void write_grid_nc(
  const char *filename,
  ctl_t * ctl,
  double *cd,
  double *vmr_expl,
  double *vmr_impl,
  double t,
  double *z,
  double *lon,
  double *lat,
  double *area,
  double dz,
  int *np) {

  double *help;

  int *help2, ncid, dimid[10], varid;

  size_t start[2], count[2];

  /* Allocate... */
  ALLOC(help, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(help2, int,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);

  /* Create file... */
  nc_create(filename, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
  NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[1]));
  NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
  NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[3]));
  NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[1], "altitude", "km");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[3], "longitude", "degrees_east");
  NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[1], "layer depth", "km");
  NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
  NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
  NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
             "volume mixing ratio (implicit)", "ppv");
  NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
             "volume mixing ratio (explicit)", "ppv");
  NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");
```

```

  /* End definitions... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", &t, 0);
  NC_PUT_DOUBLE("lon", lon, 0);
  NC_PUT_DOUBLE("lat", lat, 0);
  NC_PUT_DOUBLE("z", z, 0);
  NC_PUT_DOUBLE("area", area, 0);
  NC_PUT_DOUBLE("dz", &dz, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          cd[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("cd", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          vmr_impl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("vmr_impl", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          vmr_expl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("vmr_expl", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help2[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          np[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_INT("np", help2, 0);

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

  /* Free... */
  free(help);
  free(help2);
}

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

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

  /* Set timer... */
  SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write meteo data: %s", filename);

  /* Check compression flags... */
#ifndef ZFP
  if (ctl->met_type == 3)
    ERRMSG("zfp compression not supported!");
#endif
#ifndef ZSTD
  if (ctl->met_type == 4)
    ERRMSG("zstd compression not supported!");
#endif

  /* Write binary... */
  if (ctl->met_type >= 1 && ctl->met_type <= 4) {

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

    /* Write type of binary data... */
    FWRITE(&ctl->met_type, int,
           1,
           out);

    /* Write version of binary data... */
    int version = 100;
    FWRITE(&version, int,
```

```
	  1,
	  out);

    /* Write grid data... */
    FWRITE(&met->time, double,
	   1,
	   out);
    FWRITE(&met->nx, int,
	   1,
	   out);
    FWRITE(&met->ny, int,
	   1,
	   out);
    FWRITE(&met->np, int,
	   1,
	   out);
    FWRITE(met->lon, double,
	     (size_t) met->nx,
	   out);
    FWRITE(met->lat, double,
	     (size_t) met->ny,
	   out);
    FWRITE(met->p, double,
	     (size_t) met->np,
	   out);

    /* Write surface data... */
    write_met_bin_2d(out, met, met->ps, "PS");
    write_met_bin_2d(out, met, met->ts, "TS");
    write_met_bin_2d(out, met, met->zs, "ZS");
    write_met_bin_2d(out, met, met->us, "US");
    write_met_bin_2d(out, met, met->vs, "VS");
    write_met_bin_2d(out, met, met->pbl, "PBL");
    write_met_bin_2d(out, met, met->pt, "PT");
    write_met_bin_2d(out, met, met->tt, "TT");
    write_met_bin_2d(out, met, met->zt, "ZT");
    write_met_bin_2d(out, met, met->h2ot, "H2OT");
    write_met_bin_2d(out, met, met->pct, "PCT");
    write_met_bin_2d(out, met, met->pcb, "PCB");
    write_met_bin_2d(out, met, met->cl, "CL");
    write_met_bin_2d(out, met, met->plcl, "PLCL");
    write_met_bin_2d(out, met, met->plfc, "PLFC");
    write_met_bin_2d(out, met, met->pel, "PEL");
    write_met_bin_2d(out, met, met->cape, "CAPE");
    write_met_bin_2d(out, met, met->cin, "CIN");

    /* Write level data... */
    write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
    write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
    write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
    write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
    write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
    write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
    write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
    write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
    write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
    write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);

    /* Write final flag... */
    int final = 999;
    FWRITE(&final, int,
	   1,
	   out);

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

  return 0;
}

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

void write_met_bin_2d(
  FILE * out,
  met_t * met,
  float var[EX][EY],
  char *varname) {

  float *help;

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

  /* Copy data... */
  for (int ix = 0; ix < met->nx; ix++)
```

```
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];

  /* Write uncompressed data... */
  LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
  FWRITE(help, float,
         (size_t) (met->nx * met->ny),
         out);

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

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

void write_met_bin_3d(
  FILE * out,
  ctl_t * ctl,
  met_t * met,
  float var[EX][EY][EP],
  char *varname,
  int precision,
  double tolerance) {

  float *help;

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

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];

  /* Write uncompressed data... */
  if (ctl->met_type == 1) {
    LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
    FWRITE(help, float,
           (size_t) (met->nx * met->ny * met->np),
           out);
  }

  /* Write packed data... */
  else if (ctl->met_type == 2)
    compress_pack(varname, help, (size_t) (met->ny * met->nx),
                  (size_t) met->np, 0, out);

  /* Write zfp data... */
#ifdef ZFP
  else if (ctl->met_type == 3)
    compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
                 tolerance, 0, out);
#endif

  /* Write zstd data... */
#ifdef ZSTD
  else if (ctl->met_type == 4)
    compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
                  out);
#endif

  /* Unknown method... */
  else {
    ERRMSG("MET_TYPE not supported!");
    LOG(3, "%d %g", precision, tolerance);
  }

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

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

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

  static FILE *out;

  static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
```

```
    dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;

  static int nobs, *obscount, ip, okay;

  /* Set timer... */
  SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);

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

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

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

    /* Allocate... */
    ALLOC(lon, double,
          ctl->prof_nx);
    ALLOC(lat, double,
          ctl->prof_ny);
    ALLOC(area, double,
          ctl->prof_ny);
    ALLOC(z, double,
          ctl->prof_nz);
    ALLOC(press, double,
          ctl->prof_nz);
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create new output file... */
    LOG(1, "Write profile data: %s", 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"
            "# $11 = number of observations\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 vertical coordinates... */
    for (int iz = 0; iz < ctl->prof_nz; iz++) {
      z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
      press[iz] = P(z[iz]);
    }

    /* Set horizontal coordinates... */
    for (int ix = 0; ix < ctl->prof_nx; ix++)
      lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
    for (int iy = 0; iy < ctl->prof_ny; iy++) {
      lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
      area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
        * cos(lat[iy] * M_PI / 180.);
    }
  }

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

```
  /* Allocate... */
  ALLOC(mass, double,
        ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
  ALLOC(obsmean, double,
        ctl->prof_nx * ctl->prof_ny);
  ALLOC(obscount, int,
        ctl->prof_nx * ctl->prof_ny);

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Check observation data... */
    if (!isfinite(robs[i]))
      continue;

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

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

    /* Get mean observation index... */
    int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
    obsmean[idx] += robs[i];
    obscount[idx]++;
  }

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

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

    /* Get indices... */
    int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
    int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
    int 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... */
    int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
    mass[idx] += atm->q[ctl->qnt_m][ip];
  }

  /* Extract profiles... */
  for (int ix = 0; ix < ctl->prof_nx; ix++)
    for (int iy = 0; iy < ctl->prof_ny; iy++) {
      int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
      if (obscount[idx2] > 0) {

        /* Check profile... */
        okay = 0;
        for (int iz = 0; iz < ctl->prof_nz; iz++) {
          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
          if (mass[idx3] > 0) {
            okay = 1;
            break;
          }
        }
        if (!okay)
          continue;

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

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

          /* Get temperature, water vapor, and ozone... */
          INTPOL_INIT;
          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
                             lon[ix], lat[iy], &temp, ci, cw, 1);
          intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
                             lon[ix], lat[iy], &h2o, ci, cw, 0);
```

```
          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
                             lon[ix], lat[iy], &o3, ci, cw, 0);

          /* Calculate volume mixing ratio... */
          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
          vmr = MA / ctl->molmass * mass[idx3]
            / (RHO(press[iz], temp) * area[iy] * dz * 1e9);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %d\n",
                  t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
                  obsmean[idx2] / obscount[idx2], obscount[idx2]);
        }
      }
    }

  /* Free... */
  free(mass);
  free(obsmean);
  free(obscount);

  /* Finalize... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(lon);
    free(lat);
    free(area);
    free(z);
    free(press);
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}

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

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

  static FILE *out;

  static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;

  static int nobs;

  /* Set timer... */
  SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);

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

    /* Allocate... */
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create output file... */
    LOG(1, "Write sample data: %s", 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 = surface area [km^2]\n"
            "# $6 = layer depth [km]\n"
            "# $7 = number of particles [1]\n"
            "# $8 = column density [kg/m^2]\n"
            "# $9 = volume mixing ratio [ppv]\n"
            "# $10 = observed BT index [K]\n\n");

    /* Set latitude range, squared radius, and area... */
    dlat = DY2DEG(ctl->sample_dx);
    rmax2 = SQR(ctl->sample_dx);
    area = M_PI * rmax2;
  }

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

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Calculate Cartesian coordinates... */
    double x0[3];
    geo2cart(0, rlon[i], rlat[i], x0);

    /* Set pressure range... */
    double rp = P(rz[i]);
    double ptop = P(rz[i] + ctl->sample_dz);
    double pbot = P(rz[i] - ctl->sample_dz);

    /* Init... */
    double mass = 0;
    int np = 0;

    /* Loop over air parcels... */
#pragma omp parallel for default(shared) reduction(+:mass,np)
    for (int ip = 0; ip < atm->np; ip++) {

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

      /* Check latitude... */
      if (fabs(rlat[i] - atm->lat[ip]) > dlat)
        continue;

      /* Check horizontal distance... */
      double x1[3];
      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
      if (DIST2(x0, x1) > rmax2)
        continue;

      /* Check pressure... */
      if (ctl->sample_dz > 0)
        if (atm->p[ip] > pbot || atm->p[ip] < ptop)
          continue;

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

    /* Calculate column density... */
    double cd = mass / (1e6 * area);

    /* Calculate volume mixing ratio... */
    double vmr = 0;
    if (ctl->molmass > 0 && ctl->sample_dz > 0) {
      if (mass > 0) {

        /* Get temperature... */
        double temp;
        INTPOL_INIT;
        intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
                           rlon[i], rlat[i], &temp, ci, cw, 1);

        /* Calculate volume mixing ratio... */
        vmr = MA / ctl->molmass * mass
          / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
```

```
    }
  } else
    vmr = GSL_NAN;

  /* Write output... */
  fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
          rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
}

/* Finalize...... */
if (t == ctl->t_stop) {

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

  /* Free... */
  free(rt);
  free(rz);
  free(rlon);
  free(rlat);
  free(robs);
}
}

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

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

  static FILE *out;

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

  /* Set timer... */
  SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);

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

    /* Write info... */
    LOG(1, "Write station data: %s", 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... */
  double t0 = t - 0.5 * ctl->dt_mod;
  double 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 time range for station output... */
    if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_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);
}
```

5.23 libtrac.h File Reference

MPTRAC library declarations.

```
#include <ctype.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.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>
```

Data Structures

- struct ctl_t

  Control parameters.
- struct atm_t

  Atmospheric data.
- struct cache_t

  Cache data.
- struct clim_t

  Climatological data.
- struct met_t

  Meteo data.

Macros

- #define CPD 1003.5

  Specific heat of dry air at constant pressure [J/(kg K)].
- #define EPS (MH2O / MA)

  Ratio of the specific gas constant of dry air and water vapor [1].
- #define G0 9.80665

  Standard gravity [m/s^2].
- #define H0 7.0

  Scale height [km].
- #define LV 2501000.

  Latent heat of vaporization of water [J/kg].
- #define KB 1.3806504e-23

  Boltzmann constant [kg m^2/(K s^2)].
- #define MA 28.9644

  Molar mass of dry air [g/mol].
- #define MH2O 18.01528

  Molar mass of water vapor [g/mol].
- #define MO3 48.00

  Molar mass of ozone [g/mol].
- #define P0 1013.25

  Standard pressure [hPa].
- #define RA (1e3 ∗ RI / MA)

  Specific gas constant of dry air [J/(kg K)].
- #define RE 6367.421

  Mean radius of Earth [km].
- #define RI 8.3144598

  Ideal gas constant [J/(mol K)].
- #define T0 273.15

  Standard temperature [K].
- #define LEN 5000

  Maximum length of ASCII data lines.
- #define NP 10000000

  Maximum number of atmospheric data points.
- #define NQ 15

  Maximum number of quantities per data point.
- #define NCSI 1000000

  Maximum number of data points for CSI calculation.
- #define EP 140

  Maximum number of pressure levels for meteo data.
- #define EX 1201

  Maximum number of longitudes for meteo data.
- #define EY 601

  Maximum number of latitudes for meteo data.
- #define NENS 2000

  Maximum number of data points for ensemble analysis.
- #define NOBS 10000000

  Maximum number of observation data points.
- #define NTHREADS 512

  Maximum number of OpenMP threads.
- #define CY 250

Maximum number of latitudes for climatological data.

- #define CP 60

  Maximum number of pressure levels for climatological data.
- #define CT 12

  Maximum number of time steps for climatological data.
- #define ALLOC(ptr, type, n)

  Allocate and clear memory.
- #define ARRAY_2D(ix, iy, ny) ((ix) ∗ (ny) + (iy))

  Get 2-D array index.
- #define ARRAY_3D(ix, iy, ny, iz, nz) (((ix)∗(ny) + (iy)) ∗ (nz) + (iz))

  Get 3-D array index.
- #define DEG2DX(dlon, lat) ((dlon) ∗ M_PI ∗ RE / 180. ∗ cos((lat) / 180. ∗ M_PI))

  Convert degrees to zonal distance.
- #define DEG2DY(dlat) ((dlat) ∗ M_PI ∗ RE / 180.)

  Convert degrees to meridional distance.
- #define DP2DZ(dp, p) (- (dp) ∗ H0 / (p))

  Convert pressure change to vertical distance.
- #define DX2DEG(dx, lat)

  Convert zonal distance to degrees.
- #define DY2DEG(dy) ((dy) ∗ 180. / (M_PI ∗ RE))

  Convert meridional distance to degrees.
- #define DZ2DP(dz, p) (-(dz) ∗ (p) / H0)

  Convert vertical distance to pressure change.
- #define DIST(a, b) sqrt(DIST2(a, b))

  Compute Cartesian distance between two vectors.
- #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]))

  Compute squared distance between two vectors.
- #define DOTP(a, b) (a[0]∗b[0]+a[1]∗b[1]+a[2]∗b[2])

  Compute dot product of two vectors.
- #define FMOD(x, y) ((x) - (int) ((x) / (y)) ∗ (y))

  Compute floating point modulo.
- #define FREAD(ptr, type, size, out)

  Read binary data.
- #define FWRITE(ptr, type, size, out)

  Write binary data.
- #define INTPOL_INIT double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};

  Initialize cache variables for interpolation.
- #define INTPOL_2D(var, init)

  2-D interpolation of a meteo variable.
- #define INTPOL_3D(var, init)

  3-D interpolation of a meteo variable.
- #define INTPOL_SPACE_ALL(p, lon, lat)

  Spatial interpolation of all meteo data.
- #define INTPOL_TIME_ALL(time, p, lon, lat)

  Temporal interpolation of all meteo data.
- #define LAPSE(p1, t1, p2, t2)

  Calculate lapse rate between pressure levels.
- #define LIN(x0, y0, x1, y1, x) ((y0)+((y1)-(y0))/((x1)-(x0))∗((x)-(x0)))

  Compute linear interpolation.
- #define NC(cmd)

  Execute netCDF library command and check result.

- #define NC_DEF_VAR(varname, type, ndims, dims, long_name, units)

  Define netCDF variable.
- #define NC_GET_DOUBLE(varname, ptr, force)

  Read netCDF double array.
- #define NC_INQ_DIM(dimname, ptr, min, max)

  Read netCDF dimension.
- #define NC_PUT_DOUBLE(varname, ptr, hyperslab)

  Write netCDF double array.
- #define NC_PUT_INT(varname, ptr, hyperslab)

  Write netCDF integer array.
- #define NC_PUT_ATT(varname, attname, text)

  Set netCDF attribute.
- #define NC_PUT_ATT_GLOBAL(attname, text) NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));

  Set netCDF global attribute.
- #define NC_PUT_FLOAT(varname, ptr, hyperslab)

  Write netCDF float array.
- #define NN(x0, y0, x1, y1, x) (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))

  Compute nearest neighbor interpolation.
- #define NORM(a) sqrt(DOTP(a, a))

  Compute norm of a vector.
- #define P(z) (P0 ∗ exp(-(z) / H0))

  Convert altitude to pressure.
- #define PSAT(t) (6.112 ∗ exp(17.62 ∗ ((t) - T0) / (243.12 + (t) - T0)))

  Compute saturation pressure over water (WMO, 2018).
- #define PSICE(t) (6.112 ∗ exp(22.46 ∗ ((t) - T0) / (272.62 + (t) - T0)))

  Compute saturation pressure over ice (WMO, 2018).
- #define PW(p, h2o)

  Calculate partial water vapor pressure.
- #define RH(p, t, h2o) (PW(p, h2o) / PSAT(t) ∗ 100.)

  Compute relative humidity over water.
- #define RHICE(p, t, h2o) (PW(p, h2o) / PSICE(t) ∗ 100.)

  Compute relative humidity over ice.
- #define RHO(p, t) (100. ∗ (p) / (RA ∗ (t)))

  Compute density of air.
- #define SET_ATM(qnt, val)

  Set atmospheric quantity value.
- #define SET_QNT(qnt, name, longname, unit)

  Set atmospheric quantity index.
- #define SH(h2o) (EPS ∗ GSL_MAX((h2o), 0.1e-6))

  Compute specific humidity from water vapor volume mixing ratio.
- #define SQR(x) ((x)∗(x))

  Compute square.
- #define SWAP(x, y, type) do {type tmp = x; x = y; y = tmp;} while(0);

  Swap macro.
- #define TDEW(p, h2o)

  Calculate dew point temperature (WMO, 2018).
- #define TICE(p, h2o)

  Calculate frost point temperature (WMO, 2018).
- #define THETA(p, t) ((t) ∗ pow(1000. / (p), 0.286))

  Compute potential temperature.

- #define THETAVIRT(p, t, h2o) (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))
  Compute virtual potential temperature.
- #define TOK(line, tok, format, var)
  Get string tokens.
- #define TVIRT(t, h2o) ((t) ∗ (1. + (1. - EPS) ∗ GSL_MAX((h2o), 0.1e-6)))
  Compute virtual temperature.
- #define Z(p) (H0 ∗ log(P0 / (p)))
  Convert pressure to altitude.
- #define ZDIFF(lnp0, t0, h2o0, lnp1, t1, h2o1)
  Calculate geopotential height difference.
- #define ZETA(ps, p, t)
  Calculate zeta vertical coordinate.
- #define LOGLEV 2
  Level of log messages (0=none, 1=basic, 2=detailed, 3=debug).
- #define LOG(level, ...)
  Print log message.
- #define WARN(...)
  Print warning message.
- #define ERRMSG(...)
  Print error message and quit program.
- #define PRINT(format, var)
  Print macro for debugging.
- #define NTIMER 100
  Maximum number of timers.
- #define PRINT_TIMERS timer("END", "END", 1);
  Print timers.
- #define SELECT_TIMER(id, group, color)
  Select timer.
- #define START_TIMERS NVTX_PUSH("START", NVTX_CPU);
  Start timers.
- #define STOP_TIMERS NVTX_POP;
  Stop timers.
- #define NVTX_PUSH(range_title, range_color) {}
- #define NVTX_POP {}

Functions

- void thrustSortWrapper (double ∗__restrict__ c, int n, int ∗__restrict__ index)
  Wrapper to Thrust sorting function.
- double buoyancy_frequency (double p0, double t0, double p1, double t1)
  Calculate buoyancy frequency.
- 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 (clim_t ∗clim, double t, double lat, double p)
  Climatology of HNO3 volume mixing ratios.
- void clim_hno3_init (clim_t ∗clim)
  Initialization function for HNO3 climatology.
- double clim_oh (clim_t ∗clim, double t, double lat, double p)

Climatology of OH number concentrations.
- double clim_oh_diurnal (ctl_t ∗ctl, clim_t ∗clim, double t, double p, double lon, double lat)

  Climatology of OH number concentrations with diurnal variation.
- void clim_oh_init (ctl_t ∗ctl, clim_t ∗clim)

  Initialization function for OH climatology.
- double clim_oh_init_help (double beta, double time, double lat)

  Apply diurnal correction to OH climatology.
- double clim_h2o2 (clim_t ∗clim, double t, double lat, double p)

  Climatology of H2O2 number concentrations.
- void clim_h2o2_init (ctl_t ∗ctl, clim_t ∗clim)

  Initialization function for H2O2 climatology.
- double clim_tropo (clim_t ∗clim, double t, double lat)

  Climatology of tropopause pressure.
- void clim_tropo_init (clim_t ∗clim)

  Initialize tropopause climatology.
- void compress_pack (char ∗varname, float ∗array, size_t nxy, size_t nz, int decompress, FILE ∗inout)

  Pack or unpack array.
- void day2doy (int year, int mon, int day, int ∗doy)

  Compress or decompress array with zfp.
- 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, clim_t ∗clim, double t, met_t ∗∗met0, met_t ∗∗met1)

  Get meteo data for given time step.
- void get_met_help (ctl_t ∗ctl, double t, int direct, char ∗metbase, double dt_met, char ∗filename)

  Get meteo data for time step.
- 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 meteo 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 meteo 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)

  Spatial interpolation of meteo 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 meteo data.
- void jsec2time (double jsec, int ∗year, int ∗mon, int ∗day, int ∗hour, int ∗min, int ∗sec, double ∗remain)

  Temporal interpolation of meteo data.
- double lapse_rate (double t, double h2o)

  Calculate moist adiabatic lapse rate.
- 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.
- double nat_temperature (double p, double h2o, double hno3)

  Calculate NAT existence temperature.

- void quicksort (double arr[ ], int brr[ ], int low, int high)
  Parallel quicksort.
- int quicksort_partition (double arr[ ], int brr[ ], int low, int high)
  Partition function for quicksort.
- int read_atm (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)
  Read atmospheric data.
- int read_atm_asc (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)
  Read atmospheric data in ASCII format.
- int read_atm_bin (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)
  Read atmospheric data in binary format.
- int read_atm_clams (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)
  Read atmospheric data in CLaMS format.
- int read_atm_nc (const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)
  Read atmospheric data in netCDF format.
- void read_clim (ctl_t ∗ctl, clim_t ∗clim)
  Read climatological data.
- void read_ctl (const char ∗filename, int argc, char ∗argv[ ], ctl_t ∗ctl)
  Read control parameters.
- int read_met (char ∗filename, ctl_t ∗ctl, clim_t ∗clim, met_t ∗met)
  Read meteo data file.
- void read_met_bin_2d (FILE ∗out, met_t ∗met, float var[EX][EY], char ∗varname)
  Read 2-D meteo variable.
- void read_met_bin_3d (FILE ∗in, ctl_t ∗ctl, met_t ∗met, float var[EX][EY][EP], char ∗varname, int precision, double tolerance)
  Read 3-D meteo variable.
- void read_met_cape (clim_t ∗clim, met_t ∗met)
  Calculate convective available potential energy.
- void read_met_cloud (ctl_t ∗ctl, met_t ∗met)
  Calculate cloud properties.
- void read_met_detrend (ctl_t ∗ctl, met_t ∗met)
  Apply detrending method to temperature and winds.
- void read_met_extrapolate (met_t ∗met)
  Extrapolate meteo data at lower boundary.
- void read_met_geopot (ctl_t ∗ctl, met_t ∗met)
  Calculate geopotential heights.
- void read_met_grid (char ∗filename, int ncid, ctl_t ∗ctl, met_t ∗met)
  Read coordinates of meteo data.
- void read_met_levels (int ncid, ctl_t ∗ctl, met_t ∗met)
  Read meteo data on vertical levels.
- void read_met_ml2pl (ctl_t ∗ctl, met_t ∗met, float var[EX][EY][EP])
  Convert meteo data from model levels to pressure levels.
- int read_met_nc_2d (int ncid, char ∗varname, char ∗varname2, ctl_t ∗ctl, met_t ∗met, float dest[EX][EY], float scl, int init)
  Read and convert 2D variable from meteo data file.
- int read_met_nc_3d (int ncid, char ∗varname, char ∗varname2, ctl_t ∗ctl, met_t ∗met, float dest[EX][EY][EP], float scl, int init)
  Read and convert 3D variable from meteo data file.
- void read_met_pbl (met_t ∗met)
  Calculate pressure of the boundary layer.
- void read_met_periodic (met_t ∗met)
  Create meteo 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 meteo data.
- void read_met_surface (int ncid, met_t *met, ctl_t *ctl)

  Read surface data.
- void read_met_tropo (ctl_t *ctl, clim_t *clim, met_t *met)

  Calculate tropopause data.
- void read_obs (char *filename, double *rt, double *rz, double *rlon, double *rlat, double *robs, int *nobs)

  Read observation data.
- 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.
- double sedi (double p, double T, double rp, double rhop)

  Calculate sedimentation velocity.
- void spline (double *x, double *y, int n, double *x2, double *y2, int n2, int method)

  Spline interpolation.
- float stddev (float *data, int n)

  Calculate standard deviation.
- double sza (double sec, double lon, double lat)

  Calculate solar zenith angle.
- 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, const char *group, int output)

  Measure wall-clock time.
- double tropo_weight (clim_t *clim, double t, double lat, double p)

  Get weighting factor based on tropopause distance.
- void write_atm (const char *filename, ctl_t *ctl, atm_t *atm, double t)

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

  Write atmospheric data in ASCII format.
- void write_atm_bin (const char *filename, ctl_t *ctl, atm_t *atm)

  Write atmospheric data in binary format.
- void write_atm_clams (ctl_t *ctl, atm_t *atm, double t)

  Write atmospheric data in CLaMS format.
- void write_atm_nc (const char *filename, ctl_t *ctl, atm_t *atm)

  Write atmospheric data in netCDF format.
- 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_grid_asc (const char *filename, ctl_t *ctl, double *cd, double *vmr_expl, double *vmr_impl, double t, double *z, double *lon, double *lat, double *area, double dz, int *np)

  Write gridded data in ASCII format.
- void write_grid_nc (const char *filename, ctl_t *ctl, double *cd, double *vmr_expl, double *vmr_impl, double t, double *z, double *lon, double *lat, double *area, double dz, int *np)

  Write gridded data in netCDF format.
- int write_met (char *filename, ctl_t *ctl, met_t *met)

  Read meteo data file.

- void write_met_bin_2d (FILE ∗out, met_t ∗met, float var[EX][EY], char ∗varname)

    Write 2-D meteo variable.
- void write_met_bin_3d (FILE ∗out, ctl_t ∗ctl, met_t ∗met, float var[EX][EY][EP], char ∗varname, int precision, double tolerance)

    Write 3-D meteo variable.
- 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_sample (const char ∗filename, ctl_t ∗ctl, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double t)

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

    Write station data.

5.23.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

5.23.2 Macro Definition Documentation

5.23.2.1 CPD `#define CPD 1003.5`

Specific heat of dry air at constant pressure [J/(kg K)].

Definition at line 83 of file libtrac.h.

5.23.2.2 EPS `#define EPS (MH2O / MA)`

Ratio of the specific gas constant of dry air and water vapor [1].

Definition at line 88 of file libtrac.h.

5.23.2.3 G0 `#define G0 9.80665`

Standard gravity [m/s^2].

Definition at line 93 of file libtrac.h.

5.23.2.4 H0 `#define H0 7.0`

Scale height [km].

Definition at line 98 of file libtrac.h.

5.23.2.5 LV `#define LV 2501000.`

Latent heat of vaporization of water [J/kg].

Definition at line 103 of file libtrac.h.

5.23.2.6 KB `#define KB 1.3806504e-23`

Boltzmann constant [kg m^2/(K s^2)].

Definition at line 108 of file libtrac.h.

5.23.2.7 MA `#define MA 28.9644`

Molar mass of dry air [g/mol].

Definition at line 113 of file libtrac.h.

5.23.2.8 MH2O `#define MH2O 18.01528`

Molar mass of water vapor [g/mol].

Definition at line 118 of file libtrac.h.

5.23.2.9 MO3 `#define MO3 48.00`

Molar mass of ozone [g/mol].

Definition at line 123 of file libtrac.h.

5.23.2.10 P0 `#define P0 1013.25`

Standard pressure [hPa].

Definition at line 128 of file libtrac.h.

5.23.2.11 RA `#define RA (1e3 * RI / MA)`

Specific gas constant of dry air [J/(kg K)].

Definition at line 133 of file libtrac.h.

5.23.2.12 RE `#define RE 6367.421`

Mean radius of Earth [km].

Definition at line 138 of file libtrac.h.

5.23.2.13 RI `#define RI 8.3144598`

Ideal gas constant [J/(mol K)].

Definition at line 143 of file libtrac.h.

5.23.2.14 T0 `#define T0 273.15`

Standard temperature [K].

Definition at line 148 of file libtrac.h.

5.23.2.15 LEN `#define LEN 5000`

Maximum length of ASCII data lines.

Definition at line 157 of file libtrac.h.

5.23.2.16 NP `#define NP 10000000`

Maximum number of atmospheric data points.

Definition at line 162 of file libtrac.h.

5.23.2.17 NQ `#define NQ 15`

Maximum number of quantities per data point.

Definition at line 167 of file libtrac.h.

5.23.2.18 NCSI `#define NCSI 1000000`

Maximum number of data points for CSI calculation.

Definition at line 172 of file libtrac.h.

5.23.2.19 EP `#define EP 140`

Maximum number of pressure levels for meteo data.

Definition at line 177 of file libtrac.h.

5.23.2.20 EX `#define EX 1201`

Maximum number of longitudes for meteo data.

Definition at line 182 of file libtrac.h.

5.23.2.21 EY `#define EY 601`

Maximum number of latitudes for meteo data.

Definition at line 187 of file libtrac.h.

5.23.2.22 NENS `#define NENS 2000`

Maximum number of data points for ensemble analysis.

Definition at line 192 of file libtrac.h.

5.23.2.23 NOBS `#define NOBS 10000000`

Maximum number of observation data points.

Definition at line 197 of file libtrac.h.

5.23.2.24 NTHREADS `#define NTHREADS 512`

Maximum number of OpenMP threads.

Definition at line 202 of file libtrac.h.

5.23.2.25 CY `#define CY 250`

Maximum number of latitudes for climatological data.

Definition at line 207 of file libtrac.h.

5.23.2.26 CP `#define CP 60`

Maximum number of pressure levels for climatological data.

Definition at line 212 of file libtrac.h.

5.23.2.27 CT `#define CT 12`

Maximum number of time steps for climatological data.

Definition at line 217 of file libtrac.h.

5.23.2.28 ALLOC

```
#define ALLOC(
    ptr,
    type,
    n )
```

Value:

```
if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)        \
  ERRMSG("Out of memory!");
```

Allocate and clear memory.

Definition at line 232 of file libtrac.h.

5.23.2.29 ARRAY_2D

```
#define ARRAY_2D(
    ix,
    iy,
    ny )  ((ix) * (ny) + (iy))
```

Get 2-D array index.

Definition at line 238 of file libtrac.h.

5.23.2.30 ARRAY_3D

```
#define ARRAY_3D(
    ix,
    iy,
    ny,
    iz,
    nz )  (((ix)*(ny) + (iy)) * (nz) + (iz))
```

Get 3-D array index.

Definition at line 242 of file libtrac.h.

5.23.2.31 DEG2DX

```
#define DEG2DX(
    dlon,
    lat )  ((dlon) * M_PI * RE / 180.  * cos((lat) / 180.  * M_PI))
```

Convert degrees to zonal distance.

Definition at line 246 of file libtrac.h.

5.23.2.32 DEG2DY

```
#define DEG2DY(
    dlat )  ((dlat) * M_PI * RE / 180.)
```

Convert degrees to meridional distance.

Definition at line 250 of file libtrac.h.

5.23.2.33 DP2DZ

```
#define DP2DZ(
    dp,
    p ) (- (dp) * H0 / (p))
```

Convert pressure change to vertical distance.

Definition at line 254 of file libtrac.h.

5.23.2.34 DX2DEG

```
#define DX2DEG(
    dx,
    lat )
```

Value:

```
(((lat) < -89.999 || (lat) > 89.999) ? 0                    \
 : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))
```

Convert zonal distance to degrees.

Definition at line 258 of file libtrac.h.

5.23.2.35 DY2DEG

```
#define DY2DEG(
    dy ) ((dy) * 180. / (M_PI * RE))
```

Convert meridional distance to degrees.

Definition at line 263 of file libtrac.h.

5.23.2.36 DZ2DP

```
#define DZ2DP(
    dz,
    p ) (-(dz) * (p) / H0)
```

Convert vertical distance to pressure change.

Definition at line 267 of file libtrac.h.

5.23.2.37 DIST

```
#define DIST(
    a,
    b ) sqrt(DIST2(a, b))
```

Compute Cartesian distance between two vectors.

Definition at line 271 of file libtrac.h.

5.23.2.38 DIST2

```
#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]))
```

Compute squared distance between two vectors.

Definition at line 275 of file libtrac.h.

5.23.2.39 DOTP

```
#define DOTP(
    a,
    b ) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
```

Compute dot product of two vectors.

Definition at line 279 of file libtrac.h.

5.23.2.40 FMOD

```
#define FMOD(
    x,
    y ) ((x) - (int) ((x) / (y)) * (y))
```

Compute floating point modulo.

Definition at line 283 of file libtrac.h.

5.23.2.41 FREAD

```
#define FREAD(
    ptr,
    type,
    size,
    out )
```

Value:

```
  {                                         \
  if(fread(ptr, sizeof(type), size, out)!=size)                      \
    ERRMSG("Error while reading!");                                  \
}
```

Read binary data.

Definition at line 287 of file libtrac.h.

5.23.2.42 FWRITE

```
#define FWRITE(
    ptr,
    type,
    size,
    out )
```

Value:

```
{                                                     \
  if(fwrite(ptr, sizeof(type), size, out)!=size)                       \
    ERRMSG("Error while writing!");                                     \
}
```

Write binary data.

Definition at line 293 of file libtrac.h.

5.23.2.43 INTPOL_INIT

```
#define INTPOL_INIT double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};
```

Initialize cache variables for interpolation.

Definition at line 299 of file libtrac.h.

5.23.2.44 INTPOL_2D

```
#define INTPOL_2D(
    var,
    init )
```

Value:

```
intpol_met_time_2d(met0, met0->var, met1, met1->var,                         \
                   atm->time[ip], atm->lon[ip], atm->lat[ip],          \
                   &var, ci, cw, init);
```

2-D interpolation of a meteo variable.

Definition at line 303 of file libtrac.h.

5.23.2.45 INTPOL_3D

```
#define INTPOL_3D(
    var,
    init )
```

Value:

```
intpol_met_time_3d(met0, met0->var, met1, met1->var,                         \
                   atm->time[ip], atm->p[ip],                          \
                   atm->lon[ip], atm->lat[ip],                         \
                   &var, ci, cw, init);
```

3-D interpolation of a meteo variable.

Definition at line 309 of file libtrac.h.

5.23.2.46 INTPOL_SPACE_ALL

```
#define INTPOL_SPACE_ALL(
        p,
        lon,
        lat )
```

Value:

```
{                                                   \
intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1);         \
intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0);         \
intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0);         \
intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0);         \
intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0);         \
intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0);       \
intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0);     \
intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0);       \
intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0);     \
intpol_met_space_3d(met, met->iwc, p, 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->ts, lon, lat, &ts, ci, cw, 0);          \
intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0);          \
intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0);          \
intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0);          \
intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0);        \
intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);          \
intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0);          \
intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0);          \
intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0);      \
intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0);        \
intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0);        \
intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0);          \
intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0);      \
intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0);      \
intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0);        \
intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0);      \
intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0);        \
}
```

Spatial interpolation of all meteo data.

Definition at line 316 of file libtrac.h.

5.23.2.47 INTPOL_TIME_ALL

```
#define INTPOL_TIME_ALL(
        time,
        p,
        lon,
        lat )
```

Value:

```
{                                           \
intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0); \
intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0); \
intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0); \
intpol_met_time_2d(met0, met0->cl, met1, met1->cl, time, lon, lat, &cl, ci, cw, 0); \
intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
```

```
intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0); \
}
```

Temporal interpolation of all meteo data.

Definition at line 348 of file libtrac.h.

5.23.2.48 LAPSE

```
#define LAPSE(
    p1,
    t1,
    p2,
    t2 )
```

Value:

```
(1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1))                         \
 * ((p2) + (p1)) / ((p2) - (p1)))
```

Calculate lapse rate between pressure levels.

Definition at line 380 of file libtrac.h.

5.23.2.49 LIN

```
#define LIN(
    x0,
    y0,
    x1,
    y1,
    x )   ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
```

Compute linear interpolation.

Definition at line 385 of file libtrac.h.

5.23.2.50 NC

```
#define NC(
    cmd )
```

Value:

```
{                                          \
int nc_result=(cmd);                                          \
if(nc_result!=NC_NOERR)                                       \
  ERRMSG("%s", nc_strerror(nc_result));                       \
}
```

Execute netCDF library command and check result.

Definition at line 389 of file libtrac.h.

5.23.2.51 NC_DEF_VAR

```
#define NC_DEF_VAR(
    varname,
    type,
    ndims,
    dims,
    long_name,
    units )
```

Value:

```
{   \
  NC(nc_def_var(ncid, varname, type, ndims, dims, &varid));            \
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name)); \
  NC(nc_put_att_text(ncid, varid, "units", strlen(units), units));     \
}
```

Define netCDF variable.

Definition at line 396 of file libtrac.h.

5.23.2.52 NC_GET_DOUBLE

```
#define NC_GET_DOUBLE(
    varname,
    ptr,
    force )
```

Value:

```
{                    \
  if(force) {                                                     \
    NC(nc_inq_varid(ncid, varname, &varid));                      \
    NC(nc_get_var_double(ncid, varid, ptr));                      \
  } else {                                                        \
    if(nc_inq_varid(ncid, varname, &varid) == NC_NOERR) {         \
      NC(nc_get_var_double(ncid, varid, ptr));                    \
    } else                                                        \
      WARN("netCDF variable %s is missing!", varname);     \
  }                                                               \
}
```

Read netCDF double array.

Definition at line 403 of file libtrac.h.

5.23.2.53 NC_INQ_DIM

```
#define NC_INQ_DIM(
    dimname,
    ptr,
    min,
    max )
```

Value:

```
{            \
  int dimid; size_t naux;                                   \
  NC(nc_inq_dimid(ncid, dimname, &dimid));                  \
  NC(nc_inq_dimlen(ncid, dimid, &naux));                    \
  *ptr = (int)naux;                                         \
  if ((*ptr) < (min) || (*ptr) > (max))                     \
    ERRMSG("Dimension %s is out of range!", dimname); \
}
```

Read netCDF dimension.

Definition at line 416 of file libtrac.h.

5.23.2.54 NC_PUT_DOUBLE

```
#define NC_PUT_DOUBLE(
        varname,
        ptr,
        hyperslab )
```

Value:

```
{              \
  NC(nc_inq_varid(ncid, varname, &varid));                    \
  if(hyperslab) {                                             \
    NC(nc_put_vara_double(ncid, varid, start, count, ptr));   \
  } else {                                                    \
    NC(nc_put_var_double(ncid, varid, ptr));                  \
  }                                                           \
}
```

Write netCDF double array.

Definition at line 426 of file libtrac.h.

5.23.2.55 NC_PUT_INT

```
#define NC_PUT_INT(
        varname,
        ptr,
        hyperslab )
```

Value:

```
{                      \
  NC(nc_inq_varid(ncid, varname, &varid));                    \
  if(hyperslab) {                                             \
    NC(nc_put_vara_int(ncid, varid, start, count, ptr));      \
  } else {                                                    \
    NC(nc_put_var_int(ncid, varid, ptr));                     \
  }                                                           \
}
```

Write netCDF integer array.

Definition at line 436 of file libtrac.h.

5.23.2.56 NC_PUT_ATT

```
#define NC_PUT_ATT(
        varname,
        attname,
        text )
```

Value:

```
{                              \
  NC(nc_inq_varid(ncid, varname, &varid));                            \
  NC(nc_put_att_text(ncid, varid, attname, strlen(text), text));      \
}
```

Set netCDF attribute.

Definition at line 446 of file libtrac.h.

5.23.2.57 NC_PUT_ATT_GLOBAL

```
#define NC_PUT_ATT_GLOBAL(
    attname,
    text ) NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));
```

Set netCDF global attribute.

Definition at line 452 of file libtrac.h.

5.23.2.58 NC_PUT_FLOAT

```
#define NC_PUT_FLOAT(
    varname,
    ptr,
    hyperslab )
```

Value:

```
    {                   \
    NC(nc_inq_varid(ncid, varname, &varid));                    \
    if(hyperslab) {                                             \
      NC(nc_put_vara_float(ncid, varid, start, count, ptr));    \
    } else {                                                    \
      NC(nc_put_var_float(ncid, varid, ptr));                   \
    }                                                           \
  }
```

Write netCDF float array.

Definition at line 456 of file libtrac.h.

5.23.2.59 NN

```
#define NN(
    x0,
    y0,
    x1,
    y1,
    x ) (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))
```

Compute nearest neighbor interpolation.

Definition at line 466 of file libtrac.h.

5.23.2.60 NORM

```
#define NORM(
    a ) sqrt(DOTP(a, a))
```

Compute norm of a vector.

Definition at line 470 of file libtrac.h.

5.23.2.61 P

```
#define P(
    z )   (P0 * exp(-(z) / H0))
```

Convert altitude to pressure.

Definition at line 474 of file libtrac.h.

5.23.2.62 PSAT

```
#define PSAT(
    t )   (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))
```

Compute saturation pressure over water (WMO, 2018).

Definition at line 478 of file libtrac.h.

5.23.2.63 PSICE

```
#define PSICE(
    t )   (6.112 * exp(22.46 * ((t) - T0) / (272.62 + (t) - T0)))
```

Compute saturation pressure over ice (WMO, 2018).

Definition at line 482 of file libtrac.h.

5.23.2.64 PW

```
#define PW(
    p,
    h2o )
```

Value:

```
((p) * GSL_MAX((h2o), 0.1e-6)                        \
 / (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
```

Calculate partial water vapor pressure.

Definition at line 486 of file libtrac.h.

5.23.2.65 RH

```
#define RH(
    p,
    t,
    h2o )   (PW(p, h2o) / PSAT(t) * 100.)
```

Compute relative humidity over water.

Definition at line 491 of file libtrac.h.

5.23.2.66 RHICE

```
#define RHICE(
    p,
    t,
    h2o )   (PW(p, h2o) / PSICE(t) * 100.)
```

Compute relative humidity over ice.

Definition at line 495 of file libtrac.h.

5.23.2.67 RHO

```
#define RHO(
    p,
    t )   (100. * (p) / (RA * (t)))
```

Compute density of air.

Definition at line 499 of file libtrac.h.

5.23.2.68 SET_ATM

```
#define SET_ATM(
    qnt,
    val )
```

Value:

```
if (ctl->qnt >= 0)                          \
  atm->q[ctl->qnt][ip] = val;
```

Set atmospheric quantity value.

Definition at line 503 of file libtrac.h.

5.23.2.69 SET_QNT

```
#define SET_QNT(
    qnt,
    name,
    longname,
    unit )
```

Value:

```
if (strcasecmp(ctl->qnt_name[iq], name) == 0) {        \
  ctl->qnt = iq;                                       \
  sprintf(ctl->qnt_longname[iq], longname);            \
  sprintf(ctl->qnt_unit[iq], unit);                    \
} else
```

Set atmospheric quantity index.

Definition at line 508 of file libtrac.h.

5.23.2.70 SH

```
#define SH(
    h2o )   (EPS * GSL_MAX((h2o), 0.1e-6))
```

Compute specific humidity from water vapor volume mixing ratio.

Definition at line 516 of file libtrac.h.

5.23.2.71 SQR

```
#define SQR(
    x )   ((x)*(x))
```

Compute square.

Definition at line 520 of file libtrac.h.

5.23.2.72 SWAP

```
#define SWAP(
    x,
    y,
    type )   do {type tmp = x; x = y; y = tmp;} while(0);
```

Swap macro.

Definition at line 524 of file libtrac.h.

5.23.2.73 TDEW

```
#define TDEW(
    p,
    h2o )
```

Value:

```
(T0 + 243.12 * log(PW((p), (h2o)) / 6.112)        \
 / (17.62 - log(PW((p), (h2o)) / 6.112)))
```

Calculate dew point temperature (WMO, 2018).

Definition at line 528 of file libtrac.h.

5.23.2.74 TICE

```
#define TICE(
    p,
    h2o )
```

Value:

```
(T0 + 272.62 * log(PW((p), (h2o)) / 6.112)        \
 / (22.46 - log(PW((p), (h2o)) / 6.112)))
```

Calculate frost point temperature (WMO, 2018).

Definition at line 533 of file libtrac.h.

5.23.2.75 THETA

```
#define THETA(
        p,
        t )  ((t) * pow(1000. / (p), 0.286))
```

Compute potential temperature.

Definition at line 538 of file libtrac.h.

5.23.2.76 THETAVIRT

```
#define THETAVIRT(
        p,
        t,
        h2o )  (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))
```

Compute virtual potential temperature.

Definition at line 542 of file libtrac.h.

5.23.2.77 TOK

```
#define TOK(
        line,
        tok,
        format,
        var )
```

Value:

```
  {                                                         \
    if(((tok)=strtok((line), " \t"))) {                                               \
      if(sscanf(tok, format, &(var))!=1) continue;                                    \
    } else ERRMSG("Error while reading!");                                              \
  }
```

Get string tokens.

Definition at line 546 of file libtrac.h.

5.23.2.78 TVIRT

```
#define TVIRT(
        t,
        h2o )  ((t) * (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
```

Compute virtual temperature.

Definition at line 553 of file libtrac.h.

5.23.2.79 Z

```
#define Z(
        p )  (H0 * log(P0 / (p)))
```

Convert pressure to altitude.

Definition at line 557 of file libtrac.h.

5.23.2.80 ZDIFF `#define ZDIFF(`

```
lnp0,
t0,
h2o0,
lnp1,
t1,
h2o1 )
```

Value:

```
(RI / MA / G0 * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1)))      \
 * ((lnp0) - (lnp1)))
```

Calculate geopotential height difference.

Definition at line 561 of file libtrac.h.

5.23.2.81 ZETA `#define ZETA(`

```
ps,
p,
t )
```

Value:

```
(((p) / (ps) <= 0.3 ? 1. :                                              \
  sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3)))                      \
 * THETA((p), (t)))
```

Calculate zeta vertical coordinate.

Definition at line 566 of file libtrac.h.

5.23.2.82 LOGLEV `#define LOGLEV 2`

Level of log messages (0=none, 1=basic, 2=detailed, 3=debug).

Definition at line 577 of file libtrac.h.

5.23.2.83 LOG `#define LOG(`

```
level,
... )
```

Value:

```
  {                                   \
  if(level >= 2)                                                        \
    printf("  ");                                                       \
  if(level <= LOGLEV) {                                         \
    printf(__VA_ARGS__);                                                \
    printf("\n");                                                       \
  }                                                                     \
}
```

Print log message.

Definition at line 581 of file libtrac.h.

5.23.2.84 WARN

```
#define WARN(
        ... )
```

Value:

```
{                                                          \
  printf("\nWarning (%s, %s, l%d): ", __FILE__, __func__, __LINE__);  \
  LOG(0, __VA_ARGS__);                                                \
}
```

Print warning message.

Definition at line 591 of file libtrac.h.

5.23.2.85 ERRMSG

```
#define ERRMSG(
        ... )
```

Value:

```
{                                                          \
  printf("\nError (%s, %s, l%d): ", __FILE__, __func__, __LINE__);    \
  LOG(0, __VA_ARGS__);                                                \
  exit(EXIT_FAILURE);                                                 \
}
```

Print error message and quit program.

Definition at line 597 of file libtrac.h.

5.23.2.86 PRINT

```
#define PRINT(
        format,
        var )
```

Value:

```
printf("Print (%s, %s, l%d): %s= "format"\n",                         \
       __FILE__, __func__, __LINE__, #var, var);
```

Print macro for debugging.

Definition at line 604 of file libtrac.h.

5.23.2.87 NTIMER

```
#define NTIMER 100
```

Maximum number of timers.

Definition at line 613 of file libtrac.h.

5.23.2.88 PRINT_TIMERS

```
#define PRINT_TIMERS timer("END", "END", 1);
```

Print timers.

Definition at line 616 of file libtrac.h.

5.23.2.89 SELECT_TIMER

```
#define SELECT_TIMER(
        id,
        group,
        color )
```

Value:

```
{                              \
  NVTX_POP;                                              \
  NVTX_PUSH(id, color);                                  \
  timer(id, group, 0);                                   \
}
```

Select timer.

Definition at line 620 of file libtrac.h.

5.23.2.90 START_TIMERS

```
#define START_TIMERS NVTX_PUSH("START", NVTX_CPU);
```

Start timers.

Definition at line 627 of file libtrac.h.

5.23.2.91 STOP_TIMERS

```
#define STOP_TIMERS NVTX_POP;
```

Stop timers.

Definition at line 631 of file libtrac.h.

5.23.2.92 NVTX_PUSH

```
#define NVTX_PUSH(
        range_title,
        range_color ) {}
```

Definition at line 678 of file libtrac.h.

5.23.2.93 NVTX_POP

```
#define NVTX_POP {}
```

Definition at line 679 of file libtrac.h.

5.23.3 Function Documentation

5.23.3.1 thrustSortWrapper()

```
void thrustSortWrapper (
        double *__restrict__ c,
        int n,
        int *__restrict__ index )
```

Wrapper to Thrust sorting function.

5.23.3.2 buoyancy_frequency()

```
double buoyancy_frequency (
        double p0,
        double t0,
        double p1,
        double t1 )
```

Calculate buoyancy frequency.

Definition at line 29 of file libtrac.c.

```
                 {

  double theta0 = THETA(p0, t0);
  double theta1 = THETA(p1, t1);
  double dz = RI / MA / G0 * 0.5 * (t0 + t1) * (log(p0) - log(p1));

  return sqrt(2. * G0 / (theta0 + theta1) * (theta1 - theta0) / dz);
}
```

5.23.3.3 cart2geo()

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

Convert Cartesian coordinates to geolocation.

Definition at line 44 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;
}
```

5.23.3.4 check_finite()

```
int check_finite (
        const double x )
```

Check if x is finite.

5.23.3.5 clim_hno3()

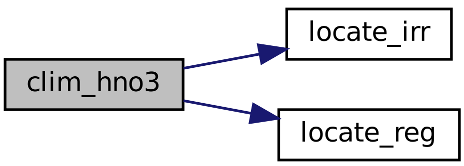
```
double clim_hno3 (
        clim_t * clim,
        double t,
        double lat,
        double p )
```

Climatology of HNO3 volume mixing ratios.

Definition at line 58 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_p[0])
    p = clim->hno3_p[0];
  else if (p > clim->hno3_p[clim->hno3_np - 1])
    p = clim->hno3_p[clim->hno3_np - 1];

  /* Check latitude... */
  if (lat < clim->hno3_lat[0])
    lat = clim->hno3_lat[0];
  else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
    lat = clim->hno3_lat[clim->hno3_nlat - 1];

  /* Get indices... */
  int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
  int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
  int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);

  /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
  double aux00 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec][ilat][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec][ilat][ip + 1], p);
  double aux01 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec][ilat + 1][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec][ilat + 1][ip + 1], p);
  double aux10 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec + 1][ilat][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec + 1][ilat][ip + 1], p);
  double aux11 = LIN(clim->hno3_p[ip],
                     clim->hno3[isec + 1][ilat + 1][ip],
                     clim->hno3_p[ip + 1],
                     clim->hno3[isec + 1][ilat + 1][ip + 1], p);
  aux00 = LIN(clim->hno3_lat[ilat], aux00,
              clim->hno3_lat[ilat + 1], aux01, lat);
  aux11 = LIN(clim->hno3_lat[ilat], aux10,
              clim->hno3_lat[ilat + 1], aux11, lat);
  aux00 = LIN(clim->hno3_time[isec], aux00,
              clim->hno3_time[isec + 1], aux11, sec);

  /* Convert from ppb to ppv... */
  return GSL_MAX(1e-9 * aux00, 0.0);
}
```

Here is the call graph for this function:



5.23.3.6 clim_hno3_init() void clim_hno3_init (clim_t * clim)

Initialization function for HNO3 climatology.

Definition at line 116 of file libtrac.c.

```
                {

  /* Write info... */
  LOG(1, "Initialize HNO3 data...");

  clim->hno3_ntime = 12;
  double hno3_time[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
  };
  memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));

  clim->hno3_nlat = 18;
  double hno3_lat[18] = {
    -85, -75, -65, -55, -45, -35, -25, -15, -5,
    5, 15, 25, 35, 45, 55, 65, 75, 85
  };
  memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));

  clim->hno3_np = 10;
  double hno3_p[10] = {
    4.64159, 6.81292, 10, 14.678, 21.5443,
    31.6228, 46.4159, 68.1292, 100, 146.78
  };
  memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));

  double hno3[12][18][10] = {
    {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
     {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
     {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
     {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
     {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
     {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
     {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
     {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
     {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
     {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
     {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
     {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
     {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
     {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
     {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
     {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
     {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
     {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
    {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
     {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
     {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
     {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
     {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
     {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
     {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
     {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
     {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
     {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
     {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
     {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
     {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
     {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
     {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
     {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
     {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
     {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
    {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
     {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
     {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
     {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
     {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
     {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
     {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
     {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
     {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
     {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
     {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
     {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
     {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
     {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
```

```
 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
{{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
{{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
{{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
{{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
{{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
```

```
     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
    {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
    {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
    {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
    {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
  };
  memcpy(clim->hno3, hno3, sizeof(clim->hno3));

  /* Get range... */
  double hno3min = 1e99, hno3max = -1e99;
  for (int it = 0; it < clim->hno3_ntime; it++)
    for (int iz = 0; iz < clim->hno3_np; iz++)
      for (int iy = 0; iy < clim->hno3_nlat; iy++) {
```

```
        hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
        hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
      }

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->hno3_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->hno3_time[0], clim->hno3_time[1],
      clim->hno3_time[clim->hno3_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->hno3_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
      Z(clim->hno3_p[clim->hno3_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
      clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
  LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->hno3_lat[0], clim->hno3_lat[1],
      clim->hno3_lat[clim->hno3_nlat - 1]);
  LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
      1e-9 * hno3max);
}
```

5.23.3.7 clim_oh()

```
double clim_oh (
        clim_t * clim,
        double t,
        double lat,
        double p )
```

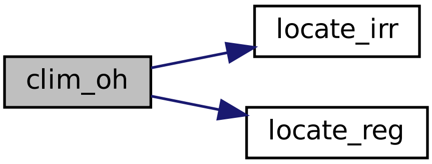
Climatology of OH number concentrations.

Definition at line 395 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_p[clim->oh_np - 1])
    p = clim->oh_p[clim->oh_np - 1];
  else if (p > clim->oh_p[0])
    p = clim->oh_p[0];

  /* Check latitude... */
  if (lat < clim->oh_lat[0])
    lat = clim->oh_lat[0];
  else if (lat > clim->oh_lat[clim->oh_nlat - 1])
    lat = clim->oh_lat[clim->oh_nlat - 1];

  /* Get indices... */
  int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
  int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
  int ip = locate_irr(clim->oh_p, clim->oh_np, p);

  /* Interpolate OH climatology... */
  double aux00 = LIN(clim->oh_p[ip],
                     clim->oh[isec][ip][ilat],
                     clim->oh_p[ip + 1],
                     clim->oh[isec][ip + 1][ilat], p);
  double aux01 = LIN(clim->oh_p[ip],
                     clim->oh[isec][ip][ilat + 1],
                     clim->oh_p[ip + 1],
                     clim->oh[isec][ip + 1][ilat + 1], p);
  double aux10 = LIN(clim->oh_p[ip],
                     clim->oh[isec + 1][ip][ilat],
                     clim->oh_p[ip + 1],
                     clim->oh[isec + 1][ip + 1][ilat], p);
  double aux11 = LIN(clim->oh_p[ip],
                     clim->oh[isec + 1][ip][ilat + 1],
                     clim->oh_p[ip + 1],
                     clim->oh[isec + 1][ip + 1][ilat + 1], p);
  aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
  aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
  aux00 =
    LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);
```

```
  return GSL_MAX(aux00, 0.0);
}
```

Here is the call graph for this function:



5.23.3.8 clim_oh_diurnal()

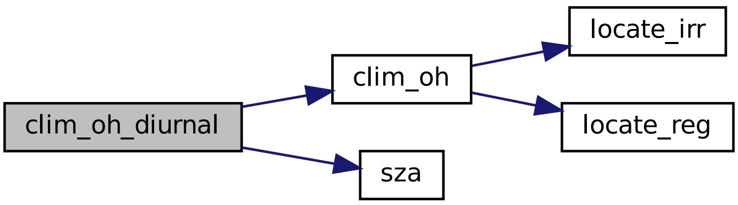
```
double clim_oh_diurnal (
        ctl_t * ctl,
        clim_t * clim,
        double t,
        double p,
        double lon,
        double lat )
```

Climatology of OH number concentrations with diurnal variation.

Definition at line 450 of file libtrac.c.

```
                    {

  double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);

  if (sza2 <= M_PI / 2. * 89. / 90.)
    return oh * exp(-ctl->oh_chem_beta / cos(sza2));
  else
    return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
}
```

Here is the call graph for this function:



5.23.3.9 clim_oh_init() void clim_oh_init (
ctl_t * ctl,
clim_t * clim)

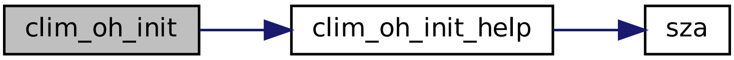
Initialization function for OH climatology.

Definition at line 468 of file libtrac.c.

```
                  {

  int nt, ncid, varid;

  double *help, ohmin = 1e99, ohmax = -1e99;

  /* Write info... */
  LOG(1, "Read OH data: %s", ctl->clim_oh_filename);

  /* Open netCDF file... */
  if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
    WARN("OH climatology data are missing!");
    return;
  }

  /* Read pressure data... */
  NC_INQ_DIM("press", &clim->oh_np, 2, CP);
  NC_GET_DOUBLE("press", clim->oh_p, 1);

  /* Check ordering of pressure data... */
  if (clim->oh_p[0] < clim->oh_p[1])
    ERRMSG("Pressure data are not descending!");

  /* Read latitudes... */
  NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
  NC_GET_DOUBLE("lat", clim->oh_lat, 1);

  /* Check ordering of latitudes... */
  if (clim->oh_lat[0] > clim->oh_lat[1])
    ERRMSG("Latitude data are not ascending!");

  /* Set time data for monthly means... */
  clim->oh_ntime = 12;
  clim->oh_time[0] = 1209600.00;
  clim->oh_time[1] = 3888000.00;
  clim->oh_time[2] = 6393600.00;
  clim->oh_time[3] = 9072000.00;
  clim->oh_time[4] = 11664000.00;
  clim->oh_time[5] = 14342400.00;
  clim->oh_time[6] = 16934400.00;
  clim->oh_time[7] = 19612800.00;
  clim->oh_time[8] = 22291200.00;
  clim->oh_time[9] = 24883200.00;
  clim->oh_time[10] = 27561600.00;
  clim->oh_time[11] = 30153600.00;

  /* Check number of timesteps... */
  NC_INQ_DIM("time", &nt, 12, 12);

  /* Read OH data... */
  ALLOC(help, double,
        clim->oh_nlat * clim->oh_np * clim->oh_ntime);
  NC_GET_DOUBLE("OH", help, 1);
  for (int it = 0; it < clim->oh_ntime; it++)
    for (int iz = 0; iz < clim->oh_np; iz++)
      for (int iy = 0; iy < clim->oh_nlat; iy++) {
        clim->oh[it][iz][iy] =
          help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
          / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
                              clim->oh_lat[iy]);
        ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
        ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
      }
  free(help);

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

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->oh_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->oh_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa",
      clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
```

```
  LOG(2, "Number of latitudes: %d", clim->oh_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
  LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
}
```

Here is the call graph for this function:



5.23.3.10 clim_oh_init_help()

```
double clim_oh_init_help (
    double beta,
    double time,
    double lat )
```

Apply diurnal correction to OH climatology.

Definition at line 555 of file libtrac.c.

```
                 {

  double aux, lon, sum = 0;

  int n = 0;

  /* Integrate day/night correction factor over longitude... */
  for (lon = -180; lon < 180; lon += 1) {
    aux = sza(time, lon, lat);
    if (aux <= M_PI / 2. * 85. / 90.)
      sum += exp(-beta / cos(aux));
    else
      sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
    n++;
  }
  return sum / (double) n;
}
```

Here is the call graph for this function:

5.23.3.11 clim_h2o2()

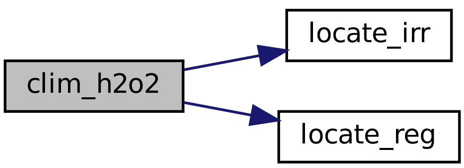
```
double clim_h2o2 (
        clim_t * clim,
        double t,
        double lat,
        double p )
```

Climatology of H2O2 number concentrations.

Definition at line 578 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->h2o2_p[clim->h2o2_np - 1])
    p = clim->h2o2_p[clim->h2o2_np - 1];
  else if (p > clim->h2o2_p[0])
    p = clim->h2o2_p[0];

  /* Check latitude... */
  if (lat < clim->h2o2_lat[0])
    lat = clim->h2o2_lat[0];
  else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
    lat = clim->h2o2_lat[clim->h2o2_nlat - 1];

  /* Get indices... */
  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);

  /* Interpolate H2O2 climatology... */
  double aux00 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec][ip][ilat],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec][ip + 1][ilat], p);
  double aux01 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec][ip][ilat + 1],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec][ip + 1][ilat + 1], p);
  double aux10 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec + 1][ip][ilat],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec + 1][ip + 1][ilat], p);
  double aux11 = LIN(clim->h2o2_p[ip],
                     clim->h2o2[isec + 1][ip][ilat + 1],
                     clim->h2o2_p[ip + 1],
                     clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
  aux00 =
    LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
  aux11 =
    LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
  aux00 =
    LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);

  return GSL_MAX(aux00, 0.0);
}
```

Here is the call graph for this function:



5.23.3.12 clim_h2o2_init()

```
void clim_h2o2_init (
    ctl_t * ctl,
    clim_t * clim )
```

Initialization function for H2O2 climatology.

Definition at line 635 of file libtrac.c.

```
                   {

  int ncid, varid, it, iy, iz, nt;

  double *help, h2o2min = 1e99, h2o2max = -1e99;

  /* Write info... */
  LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);

  /* Open netCDF file... */
  if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
    WARN("H2O2 climatology data are missing!");
    return;
  }

  /* Read pressure data... */
  NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
  NC_GET_DOUBLE("press", clim->h2o2_p, 1);

  /* Check ordering of pressure data... */
  if (clim->h2o2_p[0] < clim->h2o2_p[1])
    ERRMSG("Pressure data are not descending!");

  /* Read latitudes... */
  NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
  NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);

  /* Check ordering of latitude data... */
  if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
    ERRMSG("Latitude data are not ascending!");

  /* Set time data (for monthly means)... */
  clim->h2o2_ntime = 12;
  clim->h2o2_time[0] = 1209600.00;
  clim->h2o2_time[1] = 3888000.00;
  clim->h2o2_time[2] = 6393600.00;
  clim->h2o2_time[3] = 9072000.00;
  clim->h2o2_time[4] = 11664000.00;
  clim->h2o2_time[5] = 14342400.00;
  clim->h2o2_time[6] = 16934400.00;
  clim->h2o2_time[7] = 19612800.00;
  clim->h2o2_time[8] = 22291200.00;
  clim->h2o2_time[9] = 24883200.00;
  clim->h2o2_time[10] = 27561600.00;
  clim->h2o2_time[11] = 30153600.00;

  /* Check number of timesteps... */
  NC_INQ_DIM("time", &nt, 12, 12);

  /* Read data... */
  ALLOC(help, double,
        clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
  NC_GET_DOUBLE("h2o2", help, 1);
  for (it = 0; it < clim->h2o2_ntime; it++)
    for (iz = 0; iz < clim->h2o2_np; iz++)
      for (iy = 0; iy < clim->h2o2_nlat; iy++) {
        clim->h2o2[it][iz][iy] =
          help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
        h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
        h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
      }
  free(help);

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

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->h2o2_time[0], clim->h2o2_time[1],
      clim->h2o2_time[clim->h2o2_ntime - 1]);
  LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
  LOG(2, "Altitude levels: %g, %g ... %g km",
      Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
      Z(clim->h2o2_p[clim->h2o2_np - 1]));
  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
      clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
```

```
  LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->h2o2_lat[0], clim->h2o2_lat[1],
      clim->h2o2_lat[clim->h2o2_nlat - 1]);
  LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
}
```

5.23.3.13 clim_tropo()

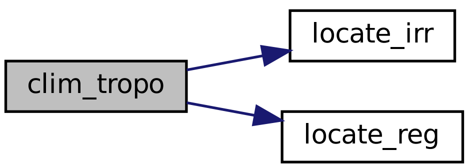
```
double clim_tropo (
            clim_t * clim,
            double t,
            double lat )
```

Climatology of tropopause pressure.

Definition at line 723 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_time, clim->tropo_ntime, sec);
  int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);

  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
  double p0 = LIN(clim->tropo_lat[ilat],
                  clim->tropo[isec][ilat],
                  clim->tropo_lat[ilat + 1],
                  clim->tropo[isec][ilat + 1], lat);
  double p1 = LIN(clim->tropo_lat[ilat],
                  clim->tropo[isec + 1][ilat],
                  clim->tropo_lat[ilat + 1],
                  clim->tropo[isec + 1][ilat + 1], lat);
  return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
}
```

Here is the call graph for this function:



5.23.3.14 clim_tropo_init()

```
void clim_tropo_init (
            clim_t * clim )
```

Initialize tropopause climatology.

Definition at line 751 of file libtrac.c.

```
                  {
```

```
  /* Write info... */
  LOG(1, "Initialize tropopause data...");

  clim->tropo_ntime = 12;
  double tropo_time[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
  };
  memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));

  clim->tropo_nlat = 73;
  double tropo_lat[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
  };
  memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));

  double tropo[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}
  };
  memcpy(clim->tropo, tropo, sizeof(clim->tropo));

  /* Get range... */
  double tropomin = 1e99, tropomax = -1e99;
  for (int it = 0; it < clim->tropo_ntime; it++)
    for (int iy = 0; iy < clim->tropo_nlat; iy++) {
      tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
      tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
    }

  /* Write info... */
  LOG(2, "Number of time steps: %d", clim->tropo_ntime);
  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
      clim->tropo_time[0], clim->tropo_time[1],
      clim->tropo_time[clim->tropo_ntime - 1]);
  LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      clim->tropo_lat[0], clim->tropo_lat[1],
      clim->tropo_lat[clim->tropo_nlat - 1]);
  LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
      Z(tropomin));
  LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
}
```

5.23.3.15 compress_pack()

```
void compress_pack (
    char * varname,
    float * array,
    size_t nxy,
    size_t nz,
    int decompress,
    FILE * inout )
```

Pack or unpack array.

Definition at line 903 of file libtrac.c.

```
                  {

  double min[EP], max[EP], off[EP], scl[EP];

```

```
  unsigned short *sarray;

  /* Allocate... */
  ALLOC(sarray, unsigned short,
        nxy * nz);

  /* Read compressed stream and decompress array... */
  if (decompress) {

    /* Write info... */
    LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
        varname, 100. * sizeof(unsigned short) / sizeof(float));

    /* Read data... */
    FREAD(&scl, double,
          nz,
          inout);
    FREAD(&off, double,
          nz,
          inout);
    FREAD(sarray, unsigned short,
          nxy * nz,
          inout);

    /* Convert to float... */
#pragma omp parallel for default(shared)
    for (size_t ixy = 0; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++)
        array[ixy * nz + iz]
          = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
  }

  /* Compress array and output compressed stream... */
  else {

    /* Write info... */
    LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
        varname, 100. * sizeof(unsigned short) / sizeof(float));

    /* Get range... */
    for (size_t iz = 0; iz < nz; iz++) {
      min[iz] = array[iz];
      max[iz] = array[iz];
    }
    for (size_t ixy = 1; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++) {
        if (array[ixy * nz + iz] < min[iz])
          min[iz] = array[ixy * nz + iz];
        if (array[ixy * nz + iz] > max[iz])
          max[iz] = array[ixy * nz + iz];
      }

    /* Get offset and scaling factor... */
    for (size_t iz = 0; iz < nz; iz++) {
      scl[iz] = (max[iz] - min[iz]) / 65533.;
      off[iz] = min[iz];
    }

    /* Convert to short... */
#pragma omp parallel for default(shared)
    for (size_t ixy = 0; ixy < nxy; ixy++)
      for (size_t iz = 0; iz < nz; iz++)
        if (scl[iz] != 0)
          sarray[ixy * nz + iz] = (unsigned short)
            ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
        else
          sarray[ixy * nz + iz] = 0;

    /* Write data... */
    FWRITE(&scl, double,
           nz,
           inout);
    FWRITE(&off, double,
           nz,
           inout);
    FWRITE(sarray, unsigned short,
           nxy * nz,
           inout);
  }

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

5.23.3.16 day2doy()

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

Compress or decompress array with zfp.

Compress or decompress array with zstd.

Get day of year from date.

Definition at line 1144 of file libtrac.c.

```
                 {

  const int
    d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
    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;
}
```

5.23.3.17 doy2day()

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

Get date from day of year.

Definition at line 1163 of file libtrac.c.

```
                 {

  const int
    d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
    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;
  }
}
```

5.23.3.18 geo2cart()

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

Convert geolocation to Cartesian coordinates.

Definition at line 1193 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);
}
```

5.23.3.19 get_met()

```
void get_met (
    ctl_t * ctl,
    clim_t * clim,
    double t,
    met_t ** met0,
    met_t ** met1 )
```

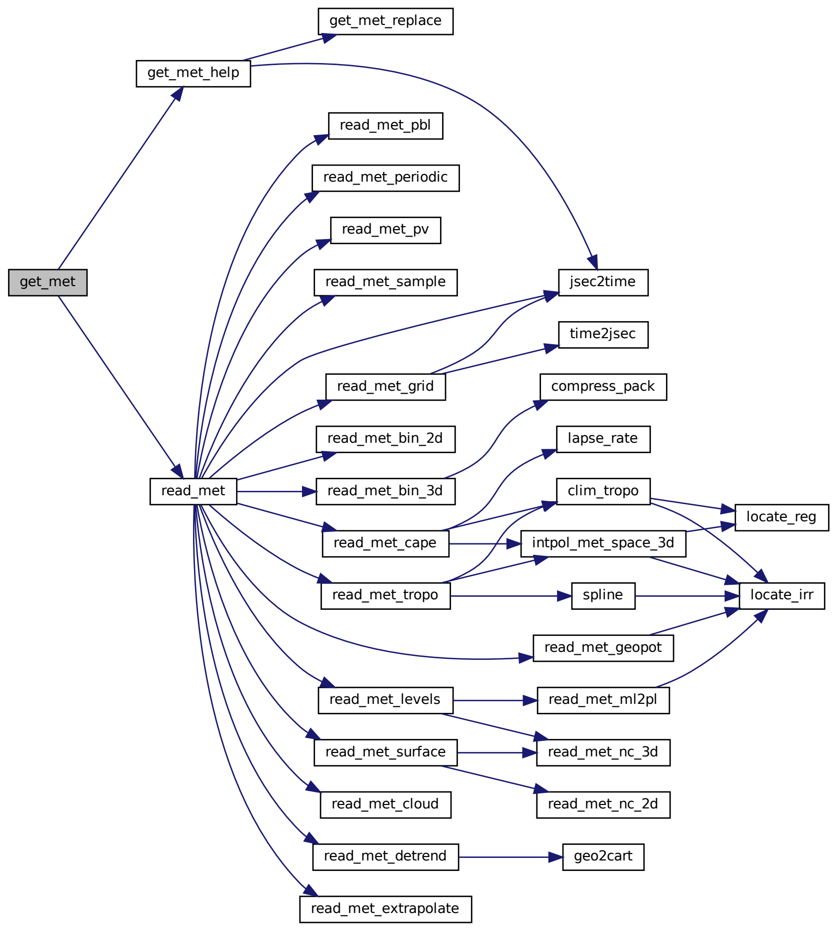
Get meteo data for given time step.

Definition at line 1207 of file libtrac.c.

```
                  {

  static int init;

  met_t *mets;

  char cachefile[LEN], cmd[2 * LEN], filename[LEN];

  /* Set timer... */
  SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);

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

    /* Read meteo data... */
    get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
                 ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met0))
      ERRMSG("Cannot open file!");

    get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
                 ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met1))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met0up = *met0;
    met_t *met1up = *met1;
#ifdef ASYNCIO
#pragma acc update device(met0up[:1],met1up[:1]) async(5)
#else
#pragma acc update device(met0up[:1],met1up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
                   ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
```

```
  }

  /* Read new data for forward trajectories... */
  if (t > (*met1)->time) {

    /* Pointer swap... */
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;

    /* Read new meteo data... */
    get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met1))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met1up = *met1;
#ifdef ASYNCIO
#pragma acc update device(met1up[:1]) async(5)
#else
#pragma acc update device(met1up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
                   cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }
  /* Read new data for backward trajectories... */
  if (t < (*met0)->time) {

    /* Pointer swap... */
    mets = *met1;
    *met1 = *met0;
    *met0 = mets;

    /* Read new meteo data... */
    get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
    if (!read_met(filename, ctl, clim, *met0))
      ERRMSG("Cannot open file!");

    /* Update GPU... */
#ifdef _OPENACC
    met_t *met0up = *met0;
#ifdef ASYNCIO
#pragma acc update device(met0up[:1]) async(5)
#else
#pragma acc update device(met0up[:1])
#endif
#endif

    /* Caching... */
    if (ctl->met_cache && t != ctl->t_stop) {
      get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
                   cachefile);
      sprintf(cmd, "cat %s > /dev/null &", cachefile);
      LOG(1, "Caching: %s", cachefile);
      if (system(cmd) != 0)
        WARN("Caching command failed!");
    }
  }
  /* Check that grids are consistent... */
  if ((*met0)->nx != 0 && (*met1)->nx != 0) {
    if ((*met0)->nx != (*met1)->nx
        || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
      ERRMSG("Meteo grid dimensions do not match!");
    for (int ix = 0; ix < (*met0)->nx; ix++)
      if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
        ERRMSG("Meteo grid longitudes do not match!");
    for (int iy = 0; iy < (*met0)->ny; iy++)
      if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
        ERRMSG("Meteo grid latitudes do not match!");
    for (int ip = 0; ip < (*met0)->np; ip++)
      if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
        ERRMSG("Meteo grid pressure levels do not match!");
  }
}
```

Here is the call graph for this function:



5.23.3.20 get_met_help()

```
void get_met_help (
        ctl_t * ctl,
        double t,
        int direct,
        char * metbase,
        double dt_met,
        char * filename )
```

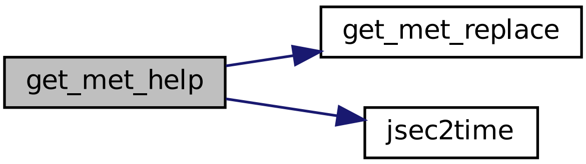
Get meteo data for time step.

Definition at line 1345 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 of MPTRAC meteo files... */
  if (ctl->clams_met_data == 0) {
    if (ctl->met_type == 0)
      sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
    else if (ctl->met_type == 1)
      sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
    else if (ctl->met_type == 2)
      sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
    else if (ctl->met_type == 3)
      sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
    else if (ctl->met_type == 4)
      sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", 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);
  }

  /* Set filename of CLaMS meteo files... */
  else {
    sprintf(filename, "%s_YYMMDDHH.nc", metbase);
    sprintf(repl, "%d", year);
    get_met_replace(filename, "YYYY", repl);
    sprintf(repl, "%d", year % 100);
    get_met_replace(filename, "YY", 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:



5.23.3.21 get_met_replace() `void get_met_replace ( char * orig, char * search, char * repl )`

Replace template strings in filename.

Definition at line 1408 of file libtrac.c.

```
                 {
```

```
  char buffer[LEN];

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

    /* Replace sub-string... */
    char *ch;
    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);
  }
}
```

5.23.3.22 intpol_met_space_3d()

```
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 )
```

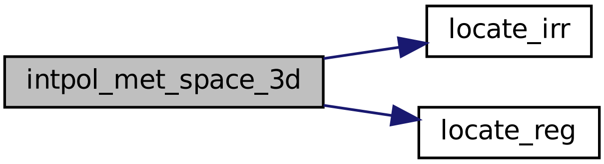
Spatial interpolation of meteo data.

Definition at line 1432 of file libtrac.c.

```
                {

  /* Initialize interpolation... */
  if (init) {

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

    /* Get interpolation indices... */
    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);

    /* Get interpolation weights... */
    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:



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

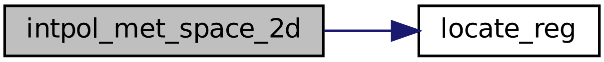
Spatial interpolation of meteo data.

Definition at line 1489 of file libtrac.c.

```
            {

  /* Initialize interpolation... */
  if (init) {

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

    /* Get interpolation indices... */
    ci[1] = locate_reg(met->lon, met->nx, lon);
    ci[2] = locate_reg(met->lat, met->ny, lat);

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

```
        *var = aux11;
      else
        *var = aux01;
    } else {
      if (cw[1] < 0.5)
        *var = aux10;
      else
        *var = aux00;
    }
  }
}
```

Here is the call graph for this function:



5.23.3.24 intpol_met_time_3d()

```
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 )
```

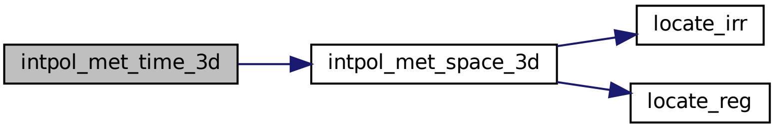
Spatial interpolation of meteo data.

Temporal interpolation of meteo data.

Definition at line 1649 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, 0);

  /* 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:



5.23.3.25 intpol_met_time_2d()

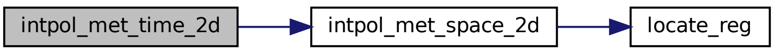
```
void intpol_met_time_2d (
    met_t * met0,
    float array0[EX][EY],
    met_t * met1,
    float array1[EX][EY],
    double ts,
    double lon,
    double lat,
    double * var,
    int * ci,
    double * cw,
    int init )
```

Temporal interpolation of meteo data.

Definition at line 1678 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, 0);

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

  /* Interpolate... */
  if (isfinite(var0) && isfinite(var1))
    *var = wt * (var0 - var1) + var1;
  else if (wt < 0.5)
    *var = var1;
  else
    *var = var0;
}
```

Here is the call graph for this function:



5.23.3.26 jsec2time()

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

Temporal interpolation of meteo data.

Convert seconds to date.

Definition at line 1742 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);
}
```

5.23.3.27 lapse_rate()

```
double lapse_rate (
    double t,
    double h2o )
```

Calculate moist adiabatic lapse rate.

Definition at line 1775 of file libtrac.c.

```
                  {

  /*
     Calculate moist adiabatic lapse rate [K/km] from temperature [K]
     and water vapor volume mixing ratio [1].

     Reference: https://en.wikipedia.org/wiki/Lapse_rate
   */

  const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));

  return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
}
```

5.23.3.28 locate_irr()

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

Find array index for irregular grid.

Definition at line 1793 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;
}
```

5.23.3.29 locate_reg()

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

Find array index for regular grid.

Definition at line 1823 of file libtrac.c.

```
                  {

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

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

5.23.3.30 nat_temperature()

```
double nat_temperature (
    double p,
    double h2o,
    double hno3 )
```

Calculate NAT existence temperature.

Definition at line 1842 of file libtrac.c.

```
                       {

  /* Check water vapor vmr... */
```

```
  h2o = GSL_MAX(h2o, 0.1e-6);

  /* Calculate T_NAT... */
  double p_hno3 = hno3 * p / 1.333224;
  double p_h2o = h2o * p / 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 tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
  if (x2 > 0)
    tnat = x2;

  return tnat;
}
```

5.23.3.31 quicksort()

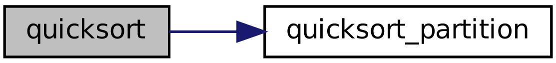
```
void quicksort (
    double arr[],
    int brr[],
    int low,
    int high )
```

Parallel quicksort.

Definition at line 1866 of file libtrac.c.

```
                 {

  if (low < high) {
    int pi = quicksort_partition(arr, brr, low, high);

#pragma omp task firstprivate(arr,brr,low,pi)
    {
      quicksort(arr, brr, low, pi - 1);
    }

    // #pragma omp task firstprivate(arr,brr,high,pi)
    {
      quicksort(arr, brr, pi + 1, high);
    }
  }
}
```

Here is the call graph for this function:



5.23.3.32 quicksort_partition()

```
int quicksort_partition (
    double arr[],
    int brr[],
    int low,
    int high )
```

Partition function for quicksort.

Definition at line 1889 of file libtrac.c.

```
                {

  double pivot = arr[high];
  int i = (low - 1);

  for (int j = low; j <= high - 1; j++)
    if (arr[j] <= pivot) {
      i++;
      SWAP(arr[i], arr[j], double);
      SWAP(brr[i], brr[j], int);
    }
  SWAP(arr[high], arr[i + 1], double);
  SWAP(brr[high], brr[i + 1], int);

  return (i + 1);
}
```

5.23.3.33 read_atm()

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

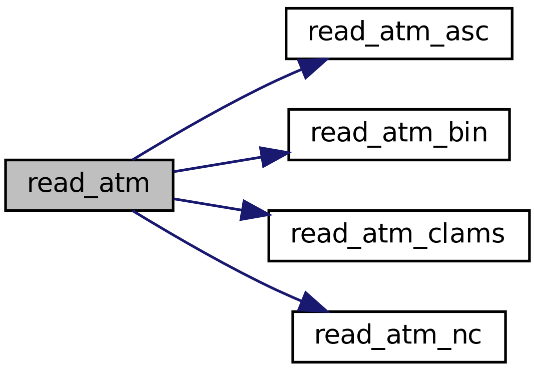
Read atmospheric data.

Definition at line 1912 of file libtrac.c.

```
                     {

  int result;

  /* Set timer... */
  SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);

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

  /* Write info... */
  LOG(1, "Read atmospheric data: %s", filename);

  /* Read ASCII data... */
  if (ctl->atm_type == 0)
    result = read_atm_asc(filename, ctl, atm);

  /* Read binary data... */
  else if (ctl->atm_type == 1)
    result = read_atm_bin(filename, ctl, atm);

  /* Read netCDF data... */
  else if (ctl->atm_type == 2)
    result = read_atm_nc(filename, ctl, atm);

  /* Read CLaMS data... */
  else if (ctl->atm_type == 3)
    result = read_atm_clams(filename, ctl, atm);

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

  /* Check result... */
  if (result != 1)
    return 0;

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of particles: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
```

```
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  for (int iq = 0; iq < ctl->nq; iq++) {
    char msg[LEN];
    sprintf(msg, "Quantity %s range: %s ... %s %s",
            ctl->qnt_name[iq], ctl->qnt_format[iq],
            ctl->qnt_format[iq], ctl->qnt_unit[iq]);
    gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
    LOG(2, msg, mini, maxi);
  }

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

Here is the call graph for this function:



5.23.3.34 read_atm_asc()

```
int read_atm_asc (
        const char * filename,
        ctl_t * ctl,
        atm_t * atm )
```

Read atmospheric data in ASCII format.

Definition at line 1983 of file libtrac.c.

```
                  {

  FILE *in;

  /* Open file... */
  if (!(in = fopen(filename, "r"))) {
    WARN("Cannot open file!");
    return 0;
  }

  /* Read line... */
  char line[LEN];
  while (fgets(line, LEN, in)) {

    /* Read data... */
    char *tok;
    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 (int 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);

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

5.23.3.35 read_atm_bin() `int read_atm_bin ( const char * filename, ctl_t * ctl, atm_t * atm )`

Read atmospheric data in binary format.

Definition at line 2026 of file libtrac.c.

```
                     {

  FILE *in;

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

  /* Check version of binary data... */
  int version;
  FREAD(&version, int,
        1,
        in);
  if (version != 100)
    ERRMSG("Wrong version of binary data!");

  /* 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 (int iq = 0; iq < ctl->nq; iq++)
    FREAD(atm->q[iq], double,
            (size_t) atm->np,
          in);

  /* Read final flag... */
  int final;
  FREAD(&final, int,
        1,
        in);
  if (final != 999)
    ERRMSG("Error while reading binary data!");

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

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

5.23.3.36 read_atm_clams()

```
int read_atm_clams (
    const char * filename,
    ctl_t * ctl,
    atm_t * atm )
```

Read atmospheric data in CLaMS format.

Definition at line 2083 of file libtrac.c.

```
                {

  int ncid, varid;

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

  /* Get dimensions... */
  NC_INQ_DIM("NPARTS", &atm->np, 1, NP);

  /* Get time... */
  if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
    NC(nc_get_var_double(ncid, varid, atm->time));
  } else {
    WARN("TIME_INIT not found use time instead!");
    double time_init;
    NC_GET_DOUBLE("time", &time_init, 1);
    for (int ip = 0; ip < atm->np; ip++) {
      atm->time[ip] = time_init;
    }
  }

  /* Read zeta coordinate, pressure is optional... */
  if (ctl->vert_coord_ap == 1) {
    NC_GET_DOUBLE("ZETA", atm->zeta, 1);
    NC_GET_DOUBLE("PRESS", atm->p, 0);
  }

  /* Read pressure, zeta coordinate is optional... */
  else {
    NC_GET_DOUBLE("PRESS", atm->p, 1);
    NC_GET_DOUBLE("ZETA", atm->zeta, 0);
  }

  /* Read longitude and latitude... */
  NC_GET_DOUBLE("LON", atm->lon, 1);
  NC_GET_DOUBLE("LAT", atm->lat, 1);

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

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

5.23.3.37 read_atm_nc()

```
int read_atm_nc (
    const char * filename,
    ctl_t * ctl,
    atm_t * atm )
```

Read atmospheric data in netCDF format.

Definition at line 2134 of file libtrac.c.

```
                {

  int ncid, varid;

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

  /* Get dimensions... */
  NC_INQ_DIM("obs", &atm->np, 1, NP);

  /* Read geolocations... */
  NC_GET_DOUBLE("time", atm->time, 1);
```

```
  NC_GET_DOUBLE("press", atm->p, 1);
  NC_GET_DOUBLE("lon", atm->lon, 1);
  NC_GET_DOUBLE("lat", atm->lat, 1);

  /* Read variables... */
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

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

5.23.3.38 read_clim()

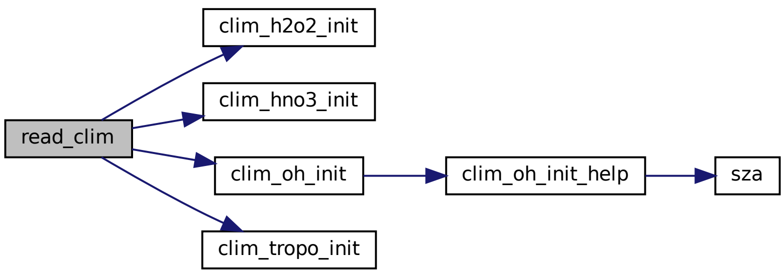
```
void read_clim (
    ctl_t * ctl,
    clim_t * clim )
```

Read climatological data.

Definition at line 2167 of file libtrac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);

  /* Init tropopause climatology... */
  clim_tropo_init(clim);

  /* Init HNO3 climatology... */
  clim_hno3_init(clim);

  /* Read OH climatology... */
  if (ctl->clim_oh_filename[0] != '-')
    clim_oh_init(ctl, clim);

  /* Read H2O2 climatology... */
  if (ctl->clim_h2o2_filename[0] != '-')
    clim_h2o2_init(ctl, clim);
}
```

Here is the call graph for this function:



5.23.3.39 read_ctl()

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

Read control parameters.

Definition at line 2191 of file libtrac.c.

```
                      {

  /* Set timer... */
  SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);

  /* Write info... */
  LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
      "(executable: %s | version: %s | compiled: %s, %s)\n",
      argv[0], VERSION, __DATE__, __TIME__);

  /* Initialize quantity indices... */
  ctl->qnt_idx = -1;
  ctl->qnt_ens = -1;
  ctl->qnt_stat = -1;
  ctl->qnt_m = -1;
  ctl->qnt_vmr = -1;
  ctl->qnt_rp = -1;
  ctl->qnt_rhop = -1;
  ctl->qnt_ps = -1;
  ctl->qnt_ts = -1;
  ctl->qnt_zs = -1;
  ctl->qnt_us = -1;
  ctl->qnt_vs = -1;
  ctl->qnt_pbl = -1;
  ctl->qnt_pt = -1;
  ctl->qnt_tt = -1;
  ctl->qnt_zt = -1;
  ctl->qnt_h2ot = -1;
  ctl->qnt_z = -1;
  ctl->qnt_p = -1;
  ctl->qnt_t = -1;
  ctl->qnt_rho = -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_pct = -1;
  ctl->qnt_pcb = -1;
  ctl->qnt_cl = -1;
  ctl->qnt_plcl = -1;
  ctl->qnt_plfc = -1;
  ctl->qnt_pel = -1;
  ctl->qnt_cape = -1;
  ctl->qnt_cin = -1;
  ctl->qnt_hno3 = -1;
  ctl->qnt_oh = -1;
  ctl->qnt_vmrimpl = -1;
  ctl->qnt_mloss_oh = -1;
  ctl->qnt_mloss_h2o2 = -1;
  ctl->qnt_mloss_wet = -1;
  ctl->qnt_mloss_dry = -1;
  ctl->qnt_mloss_decay = -1;
  ctl->qnt_psat = -1;
  ctl->qnt_psice = -1;
  ctl->qnt_pw = -1;
  ctl->qnt_sh = -1;
  ctl->qnt_rh = -1;
  ctl->qnt_rhice = -1;
  ctl->qnt_theta = -1;
  ctl->qnt_zeta = -1;
  ctl->qnt_tvirt = -1;
  ctl->qnt_lapse = -1;
  ctl->qnt_vh = -1;
  ctl->qnt_vz = -1;
  ctl->qnt_pv = -1;
  ctl->qnt_tdew = -1;
  ctl->qnt_tice = -1;
  ctl->qnt_tsts = -1;
  ctl->qnt_tnat = -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_LONGNAME", iq, ctl->qnt_name[iq],
           ctl->qnt_longname[iq]);
  scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
           ctl->qnt_format[iq]);

  /* Try to identify quantity... */
  SET_QNT(qnt_idx, "idx", "particle index", "-")
    SET_QNT(qnt_ens, "ens", "ensemble index", "-")
    SET_QNT(qnt_stat, "stat", "station flag", "-")
    SET_QNT(qnt_m, "m", "mass", "kg")
    SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
    SET_QNT(qnt_rp, "rp", "particle radius", "microns")
    SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
    SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
    SET_QNT(qnt_ts, "ts", "surface temperature", "K")
    SET_QNT(qnt_zs, "zs", "surface height", "km")
    SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
    SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
    SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
    SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
    SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
    SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
    SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
    SET_QNT(qnt_z, "z", "geopotential height", "km")
    SET_QNT(qnt_p, "p", "pressure", "hPa")
    SET_QNT(qnt_t, "t", "temperature", "K")
    SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
    SET_QNT(qnt_u, "u", "zonal wind", "m/s")
    SET_QNT(qnt_v, "v", "meridional wind", "m/s")
    SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
    SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
    SET_QNT(qnt_o3, "o3", "ozone", "ppv")
    SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
    SET_QNT(qnt_iwc, "iwc", "cloud liquid water content", "kg/kg")
    SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
    SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
    SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
    SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
    SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
    SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
    SET_QNT(qnt_cape, "cape", "convective available potential energy",
            "J/kg")
    SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
    SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
    SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
    SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
    SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
    SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
            "kg")
    SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
            "kg")
    SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
            "kg")
    SET_QNT(qnt_mloss_decay, "mloss_decay",
            "mass loss due to exponential decay", "kg")
    SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
    SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
    SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
    SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
    SET_QNT(qnt_rh, "rh", "relative humidity", "%%")
    SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%%")
    SET_QNT(qnt_theta, "theta", "potential temperature", "K")
    SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
    SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
    SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
    SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
    SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
    SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
    SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
    SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
    SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
    SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
    scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
}

/* netCDF I/O parameters... */
ctl->chunkszhint =
  (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
                    NULL);
ctl->read_mode =
```

```
    (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);

  /* Vertical coordinates and velocities... */
  ctl->vert_coord_ap =
    (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
  ctl->vert_coord_met =
    (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
  ctl->vert_vel =
    (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
  ctl->clams_met_data =
    (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);

  /* 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, "180", NULL);

  /* Meteo data... */
  scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
  ctl->met_type =
    (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
  ctl->met_nc_scale =
    (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", 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);
  if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
    ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
  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);
  if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
    ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
  ctl->met_detrend =
    scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", 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_geopot_sx
    = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
  ctl->met_geopot_sy
    = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
  ctl->met_relhum
    = (int) scan_ctl(filename, argc, argv, "MET_RELHUM", -1, "0", NULL);
  ctl->met_tropo =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
  if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
    ERRMSG("Set MET_TROPO = 0 ... 5!");
  ctl->met_tropo_lapse =
    scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
  ctl->met_tropo_nlev =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
  ctl->met_tropo_lapse_sep =
    scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
  ctl->met_tropo_nlev_sep =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
                   NULL);
  ctl->met_tropo_pv =
    scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
  ctl->met_tropo_theta =
    scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
  ctl->met_tropo_spline =
    (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
  ctl->met_cloud =
    (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
  if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
    ERRMSG("Set MET_CLOUD = 0 ... 3!");
  ctl->met_cloud_min =
    scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
  ctl->met_dt_out =
    scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
  ctl->met_cache =
    (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);

  /* Sorting... */
  ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);

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

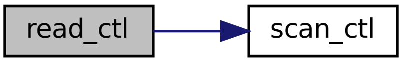
```
/* Advection parameters... */
ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
  ERRMSG("Set ADVECT to 1, 2, or 4!");
ctl->reflect =
  (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);

/* 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);

/* Convection... */
ctl->conv_cape
  = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
ctl->conv_cin
  = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
ctl->conv_mix
  = (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "1", NULL);
ctl->conv_mix_bot
  = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
ctl->conv_mix_top
  = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);

/* Boundary conditions... */
ctl->bound_mass =
  scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
ctl->bound_mass_trend =
  scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
ctl->bound_vmr =
  scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
ctl->bound_vmr_trend =
  scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
ctl->bound_lat0 =
  scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
ctl->bound_lat1 =
  scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
ctl->bound_p0 =
  scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
ctl->bound_p1 =
  scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
ctl->bound_dps =
  scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
ctl->bound_dzs =
  scan_ctl(filename, argc, argv, "BOUND_DZS", -1, "-999", NULL);
ctl->bound_zetas =
  scan_ctl(filename, argc, argv, "BOUND_ZETAS", -1, "-999", NULL);
ctl->bound_pbl =
  (int) scan_ctl(filename, argc, argv, "BOUND_PBL", -1, "0", NULL);

/* Species parameters... */
scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
  ctl->molmass = 120.907;
  ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
  ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
} else if (strcasecmp(ctl->species, "CFCl3") == 0) {
  ctl->molmass = 137.359;
  ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
  ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
} else if (strcasecmp(ctl->species, "CH4") == 0) {
  ctl->molmass = 16.043;
  ctl->oh_chem_reaction = 2;
  ctl->oh_chem[0] = 2.45e-12;
  ctl->oh_chem[1] = 1775;
  ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
  ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
} else if (strcasecmp(ctl->species, "CO") == 0) {
  ctl->molmass = 28.01;
  ctl->oh_chem_reaction = 3;
  ctl->oh_chem[0] = 6.9e-33;
  ctl->oh_chem[1] = 2.1;
  ctl->oh_chem[2] = 1.1e-12;
  ctl->oh_chem[3] = -1.3;
  ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
  ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
```

```
  } else if (strcasecmp(ctl->species, "CO2") == 0) {
    ctl->molmass = 44.009;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
  } else if (strcasecmp(ctl->species, "H2O") == 0) {
    ctl->molmass = 18.01528;
  } else if (strcasecmp(ctl->species, "N2O") == 0) {
    ctl->molmass = 44.013;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
  } else if (strcasecmp(ctl->species, "NH3") == 0) {
    ctl->molmass = 17.031;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 1.7e-12;
    ctl->oh_chem[1] = 710;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
  } else if (strcasecmp(ctl->species, "HNO3") == 0) {
    ctl->molmass = 63.012;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
  } else if (strcasecmp(ctl->species, "NO") == 0) {
    ctl->molmass = 30.006;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 7.1e-31;
    ctl->oh_chem[1] = 2.6;
    ctl->oh_chem[2] = 3.6e-11;
    ctl->oh_chem[3] = 0.1;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
  } else if (strcasecmp(ctl->species, "NO2") == 0) {
    ctl->molmass = 46.005;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 1.8e-30;
    ctl->oh_chem[1] = 3.0;
    ctl->oh_chem[2] = 2.8e-11;
    ctl->oh_chem[3] = 0.0;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
  } else if (strcasecmp(ctl->species, "O3") == 0) {
    ctl->molmass = 47.997;
    ctl->oh_chem_reaction = 2;
    ctl->oh_chem[0] = 1.7e-12;
    ctl->oh_chem[1] = 940;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
  } else if (strcasecmp(ctl->species, "SF6") == 0) {
    ctl->molmass = 146.048;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
  } else if (strcasecmp(ctl->species, "SO2") == 0) {
    ctl->molmass = 64.066;
    ctl->oh_chem_reaction = 3;
    ctl->oh_chem[0] = 2.9e-31;
    ctl->oh_chem[1] = 4.1;
    ctl->oh_chem[2] = 1.7e-12;
    ctl->oh_chem[3] = -0.2;
    ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
    ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
  } else {
    ctl->molmass =
      scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
    ctl->oh_chem_reaction =
      (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
    ctl->h2o2_chem_reaction =
      (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
                     NULL);
    for (int ip = 0; ip < 4; ip++)
      ctl->oh_chem[ip] =
        scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
    ctl->dry_depo_vdep =
      scan_ctl(filename, argc, argv, "DRY_DEPO_VDEP", -1, "0", NULL);
    ctl->dry_depo_dp =
      scan_ctl(filename, argc, argv, "DRY_DEPO_DP", -1, "30", NULL);
    ctl->wet_depo_ic_a =
      scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
    ctl->wet_depo_ic_b =
      scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
    ctl->wet_depo_bc_a =
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
    ctl->wet_depo_bc_b =
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
    for (int ip = 0; ip < 3; ip++)
      ctl->wet_depo_ic_h[ip] =
        scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
    for (int ip = 0; ip < 1; ip++)
      ctl->wet_depo_bc_h[ip] =
```

```
      scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
  }

  /* Wet deposition... */
  ctl->wet_depo_pre[0] =
    scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
  ctl->wet_depo_pre[1] =
    scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
  ctl->wet_depo_ic_ret_ratio =
    scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
  ctl->wet_depo_bc_ret_ratio =
    scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);

  /* OH chemistry... */
  ctl->oh_chem_beta =
    scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
  scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
           "../../data/clams_radical_species.nc", ctl->clim_oh_filename);

  /* H2O2 chemistry... */
  ctl->h2o2_chem_cc =
    scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
  scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
           "../../data/cams_H2O2.nc", ctl->clim_h2o2_filename);

  /* Chemistry grid... */
  ctl->chemgrid_z0 =
    scan_ctl(filename, argc, argv, "CHEMGRID_Z0", -1, "0", NULL);
  ctl->chemgrid_z1 =
    scan_ctl(filename, argc, argv, "CHEMGRID_Z1", -1, "100", NULL);
  ctl->chemgrid_nz =
    (int) scan_ctl(filename, argc, argv, "CHEMGRID_NZ", -1, "1", NULL);
  ctl->chemgrid_lon0 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LON0", -1, "-180", NULL);
  ctl->chemgrid_lon1 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LON1", -1, "180", NULL);
  ctl->chemgrid_nx =
    (int) scan_ctl(filename, argc, argv, "CHEMGRID_NX", -1, "360", NULL);
  ctl->chemgrid_lat0 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LAT0", -1, "-90", NULL);
  ctl->chemgrid_lat1 =
    scan_ctl(filename, argc, argv, "CHEMGRID_LAT1", -1, "90", NULL);
  ctl->chemgrid_ny =
    (int) scan_ctl(filename, argc, argv, "CHEMGRID_NY", -1, "180", NULL);

  /* Exponential decay... */
  ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
  ctl->tdec_strat =
    scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);

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

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

  /* 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);
  ctl->grid_type =
    (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", 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 sample data... */
  scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
           ctl->sample_basename);
  scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
           ctl->sample_obsfile);
  ctl->sample_dx =
    scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
  ctl->sample_dz =
    scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", 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);
  ctl->stat_t0 =
    scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
  ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
}
```

Here is the call graph for this function:



5.23.3.40 read_met()

```
int read_met (
    char * filename,
    ctl_t * ctl,
    clim_t * clim,
    met_t * met )
```

Read meteo data file.

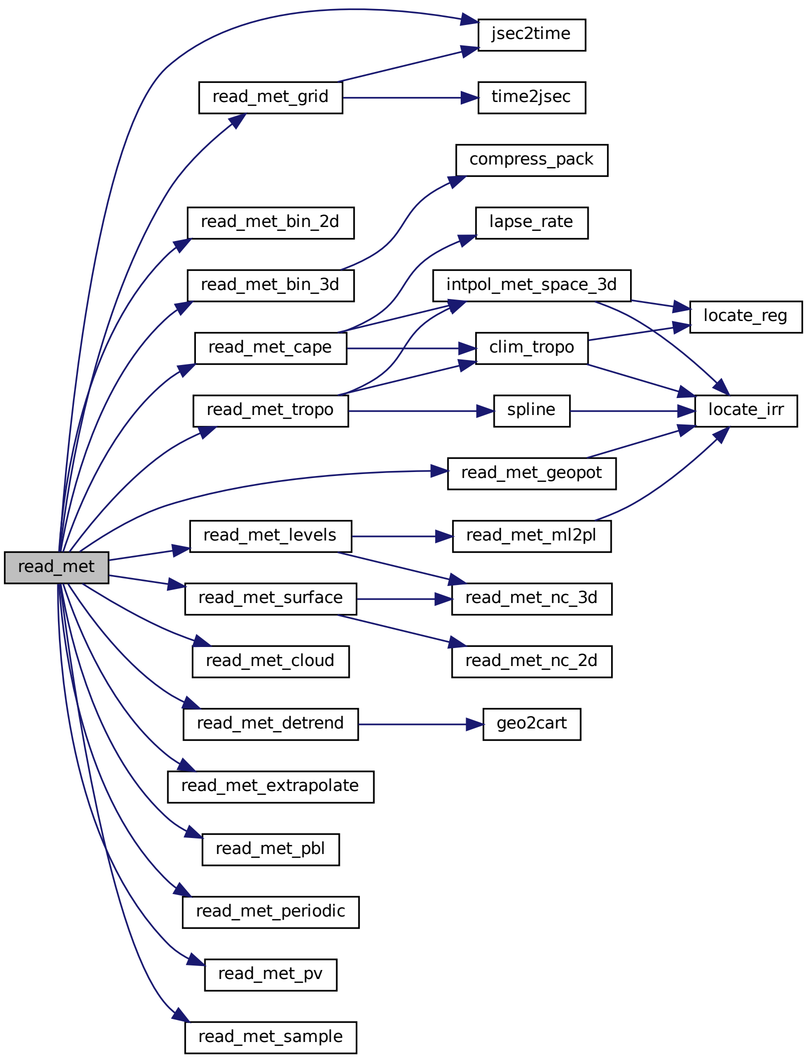
Definition at line 2782 of file libtrac.c.

```
                 {

  /* Write info... */
  LOG(1, "Read meteo data: %s", filename);

  /* Read netCDF data... */
  if (ctl->met_type == 0) {

    int ncid;

    /* Open netCDF file... */
    if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
        NC_NOERR) {
      WARN("Cannot open file!");
      return 0;
    }

    /* Read coordinates of meteo data... */
    read_met_grid(filename, ncid, ctl, met);

    /* Read meteo data on vertical levels... */
    read_met_levels(ncid, ctl, met);

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

    /* Read surface data... */
    read_met_surface(ncid, met, ctl);

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

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

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

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

    /* Calculate boundary layer data... */
    read_met_pbl(met);

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

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

```

    /* Calculate convective available potential energy... */
    read_met_cape(clim, met);

    /* Detrending... */
    read_met_detrend(ctl, met);

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

  /* Read binary data... */
  else if (ctl->met_type >= 1 && ctl->met_type <= 4) {

    FILE *in;

    double r;

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

    /* Set timer... */
    SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);

    /* Open file... */
    if (!(in = fopen(filename, "r"))) {
      WARN("Cannot open file!");
      return 0;
    }

    /* Check type of binary data... */
    int met_type;
    FREAD(&met_type, int,
          1,
          in);
    if (met_type != ctl->met_type)
      ERRMSG("Wrong MET_TYPE of binary data!");

    /* Check version of binary data... */
    int version;
    FREAD(&version, int,
          1,
          in);
    if (version != 100)
      ERRMSG("Wrong version of binary data!");

    /* Read time... */
    FREAD(&met->time, double,
          1,
          in);
    jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
    LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
        met->time, year, mon, day, hour, min);
    if (year < 1900 || year > 2100 || mon < 1 || mon > 12
        || day < 1 || day > 31 || hour < 0 || hour > 23)
      ERRMSG("Error while reading time!");

    /* Read dimensions... */
    FREAD(&met->nx, int,
          1,
          in);
    LOG(2, "Number of longitudes: %d", met->nx);
    if (met->nx < 2 || met->nx > EX)
      ERRMSG("Number of longitudes out of range!");

    FREAD(&met->ny, int,
          1,
          in);
    LOG(2, "Number of latitudes: %d", met->ny);
    if (met->ny < 2 || met->ny > EY)
      ERRMSG("Number of latitudes out of range!");

    FREAD(&met->np, int,
          1,
          in);
    LOG(2, "Number of levels: %d", met->np);
    if (met->np < 2 || met->np > EP)
      ERRMSG("Number of levels out of range!");

    /* Read grid... */
    FREAD(met->lon, double,
            (size_t) met->nx,
          in);
    LOG(2, "Longitudes: %g, %g ... %g deg",
        met->lon[0], met->lon[1], met->lon[met->nx - 1]);

    FREAD(met->lat, double,
            (size_t) met->ny,
```

```
          in);
    LOG(2, "Latitudes: %g, %g ... %g deg",
        met->lat[0], met->lat[1], met->lat[met->ny - 1]);

    FREAD(met->p, double,
          (size_t) met->np,
          in);
    LOG(2, "Altitude levels: %g, %g ... %g km",
        Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
    LOG(2, "Pressure levels: %g, %g ... %g hPa",
        met->p[0], met->p[1], met->p[met->np - 1]);

    /* Read surface data... */
    read_met_bin_2d(in, met, met->ps, "PS");
    read_met_bin_2d(in, met, met->ts, "TS");
    read_met_bin_2d(in, met, met->zs, "ZS");
    read_met_bin_2d(in, met, met->us, "US");
    read_met_bin_2d(in, met, met->vs, "VS");
    read_met_bin_2d(in, met, met->pbl, "PBL");
    read_met_bin_2d(in, met, met->pt, "PT");
    read_met_bin_2d(in, met, met->tt, "TT");
    read_met_bin_2d(in, met, met->zt, "ZT");
    read_met_bin_2d(in, met, met->h2ot, "H2OT");
    read_met_bin_2d(in, met, met->pct, "PCT");
    read_met_bin_2d(in, met, met->pcb, "PCB");
    read_met_bin_2d(in, met, met->cl, "CL");
    read_met_bin_2d(in, met, met->plcl, "PLCL");
    read_met_bin_2d(in, met, met->plfc, "PLFC");
    read_met_bin_2d(in, met, met->pel, "PEL");
    read_met_bin_2d(in, met, met->cape, "CAPE");
    read_met_bin_2d(in, met, met->cin, "CIN");

    /* Read level data... */
    read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
    read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
    read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
    read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
    read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
    read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
    read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
    read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
    read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
    read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);

    /* Read final flag... */
    int final;
    FREAD(&final, int,
          1,
          in);
    if (final != 999)
      ERRMSG("Error while reading binary data!");

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

  /* Not implemented... */
  else
    ERRMSG("MET_TYPE not implemented!");

  /* Copy wind data to cache... */
#ifdef UVW
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++) {
        met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
        met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
        met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
      }
#endif

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

Here is the call graph for this function:



5.23.3.41 read_met_bin_2d()

```
void read_met_bin_2d (
    FILE * out,
    met_t * met,
    float var[EX][EY],
    char * varname )
```

Read 2-D meteo variable.

Definition at line 3000 of file libtrac.c.

```
                      {

  float *help;

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

```
  /* Read uncompressed... */
  LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
  FREAD(help, float,
        (size_t) (met->nx * met->ny),
        in);

  /* Copy data... */
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];

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

5.23.3.42 read_met_bin_3d()

```
void read_met_bin_3d (
    FILE * in,
    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP],
    char * varname,
    int precision,
    double tolerance )
```

Read 3-D meteo variable.

Definition at line 3029 of file libtrac.c.

```
                  {

  float *help;

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

  /* Read uncompressed data... */
  if (ctl->met_type == 1) {
    LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
    FREAD(help, float,
          (size_t) (met->nx * met->ny * met->np),
          in);
  }

  /* Read packed data... */
  else if (ctl->met_type == 2)
    compress_pack(varname, help, (size_t) (met->ny * met->nx),
                  (size_t) met->np, 1, in);

  /* Read zfp data... */
  else if (ctl->met_type == 3) {
#ifdef ZFP
    compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
                 tolerance, 1, in);
#else
    ERRMSG("zfp compression not supported!");
    LOG(3, "%d %g", precision, tolerance);
#endif
  }

  /* Read zstd data... */
  else if (ctl->met_type == 4) {
#ifdef ZSTD
    compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
                  in);
#else
    ERRMSG("zstd compression not supported!");
#endif
  }

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];
```

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

Here is the call graph for this function:

5.23.3.43 read_met_cape()

void read_met_cape (
    clim_t * *clim*,
    met_t * *met*)

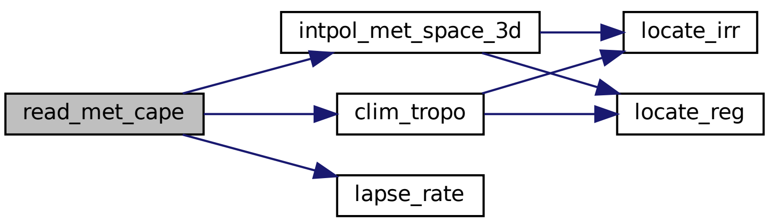
Calculate convective available potential energy.

Definition at line 3091 of file libtrac.c.

```
                 {

  /* Set timer... */
  SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
  LOG(2, "Calculate CAPE...");

  /* Vertical spacing (about 100 m)... */
  const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
      int n = 0;
      double h2o = 0, t, theta = 0;
      double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
      double ptop = pbot - 50.;
      for (int ip = 0; ip < met->np; ip++) {
        if (met->p[ip] <= pbot) {
          theta += THETA(met->p[ip], met->t[ix][iy][ip]);
          h2o += met->h2o[ix][iy][ip];
          n++;
        }
        if (met->p[ip] < ptop && n > 0)
          break;
      }
      theta /= n;
      h2o /= n;

      /* Cannot compute anything if water vapor is missing... */
      met->plcl[ix][iy] = GSL_NAN;
      met->plfc[ix][iy] = GSL_NAN;
      met->pel[ix][iy] = GSL_NAN;
      met->cape[ix][iy] = GSL_NAN;
      met->cin[ix][iy] = GSL_NAN;
      if (h2o <= 0)
        continue;

      /* Find lifted condensation level (LCL)... */
      ptop = P(20.);
      pbot = met->ps[ix][iy];
      do {
        met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
        t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
        if (RH(met->plcl[ix][iy], t, h2o) > 100.)
```

```
        ptop = met->plcl[ix][iy];
      else
        pbot = met->plcl[ix][iy];
    } while (pbot - ptop > 0.1);

    /* Calculate CIN up to LCL... */
    INTPOL_INIT;
    double dcape, dz, h2o_env, t_env;
    double p = met->ps[ix][iy];
    met->cape[ix][iy] = met->cin[ix][iy] = 0;
    do {
      dz = dz0 * TVIRT(t, h2o);
      p /= pfac;
      t = theta / pow(1000. / p, 0.286);
      intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
                          &t_env, ci, cw, 1);
      intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
                          &h2o_env, ci, cw, 0);
      dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
        TVIRT(t_env, h2o_env) * dz;
      if (dcape < 0)
        met->cin[ix][iy] += fabsf((float) dcape);
    } while (p > met->plcl[ix][iy]);

    /* Calculate level of free convection (LFC), equilibrium level (EL),
       and convective available potential energy (CAPE)... */
    dcape = 0;
    p = met->plcl[ix][iy];
    t = theta / pow(1000. / p, 0.286);
    ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
    do {
      dz = dz0 * TVIRT(t, h2o);
      p /= pfac;
      t -= lapse_rate(t, h2o) * dz;
      double psat = PSAT(t);
      h2o = psat / (p - (1. - EPS) * psat);
      intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
                          &t_env, ci, cw, 1);
      intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
                          &h2o_env, ci, cw, 0);
      double dcape_old = dcape;
      dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
        TVIRT(t_env, h2o_env) * dz;
      if (dcape > 0) {
        met->cape[ix][iy] += (float) dcape;
        if (!isfinite(met->plfc[ix][iy]))
          met->plfc[ix][iy] = (float) p;
      } else if (dcape_old > 0)
        met->pel[ix][iy] = (float) p;
      if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
        met->cin[ix][iy] += fabsf((float) dcape);
    } while (p > ptop);

    /* Check results... */
    if (!isfinite(met->plfc[ix][iy]))
      met->cin[ix][iy] = GSL_NAN;
  }
}
```

Here is the call graph for this function:



5.23.3.44 read_met_cloud()

void read_met_cloud (
    ctl_t * ctl,
    met_t * met)

Calculate cloud properties.

Definition at line 3201 of file libtrac.c.

```
                    {

  /* Set timer... */
  SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
  LOG(2, "Calculate cloud data...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

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

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

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

        /* Check ice water and liquid water content... */
        if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
            || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {

          /* Get cloud top pressure ... */
          met->pct[ix][iy]
            = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));

          /* Get cloud bottom pressure ... */
          if (!isfinite(met->pcb[ix][iy]))
            met->pcb[ix][iy]
              = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
        }

        /* 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);
      }
    }
}
```

5.23.3.45 read_met_detrend()

void read_met_detrend (
    ctl_t * ctl,
    met_t * met)

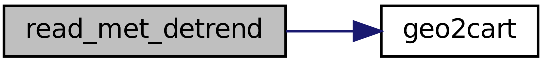
Apply detrending method to temperature and winds.

Definition at line 3251 of file libtrac.c.

```
                    {

  met_t *help;

  /* Check parameters... */
  if (ctl->met_detrend <= 0)
    return;

  /* Set timer... */
  SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
  LOG(2, "Detrend meteo data...");

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

  /* Calculate standard deviation... */
```

```
  double sigma = ctl->met_detrend / 2.355;
  double tssq = 2. * SQR(sigma);

  /* Calculate box size in latitude... */
  int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
  sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);

  /* Calculate background... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++) {
    for (int iy = 0; iy < met->ny; iy++) {

      /* Calculate Cartesian coordinates... */
      double x0[3];
      geo2cart(0.0, met->lon[ix], met->lat[iy], x0);

      /* Calculate box size in longitude... */
      int sx =
        (int) (3. * DX2DEG(sigma, met->lat[iy]) /
               fabs(met->lon[1] - met->lon[0]));
      sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);

      /* Init... */
      float wsum = 0;
      for (int ip = 0; ip < met->np; ip++) {
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
      }

      /* Loop over neighboring grid points... */
      for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
        int ix3 = ix2;
        if (ix3 < 0)
          ix3 += met->nx;
        else if (ix3 >= met->nx)
          ix3 -= met->nx;
        for (int iy2 = GSL_MAX(iy - sy, 0);
             iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {

          /* Calculate Cartesian coordinates... */
          double x1[3];
          geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);

          /* Calculate weighting factor... */
          float w = (float) exp(-DIST2(x0, x1) / tssq);

          /* Add data... */
          wsum += w;
          for (int ip = 0; ip < met->np; ip++) {
            help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
            help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
            help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
            help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
          }
        }
      }

      /* Normalize... */
      for (int ip = 0; ip < met->np; ip++) {
        help->t[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Subtract background... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++) {
        met->t[ix][iy][ip] -= help->t[ix][iy][ip];
        met->u[ix][iy][ip] -= help->u[ix][iy][ip];
        met->v[ix][iy][ip] -= help->v[ix][iy][ip];
        met->w[ix][iy][ip] -= help->w[ix][iy][ip];
      }

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

Here is the call graph for this function:



5.23.3.46 read_met_extrapolate()

```
void read_met_extrapolate (
    met_t * met )
```

Extrapolate meteo data at lower boundary.

Definition at line 3355 of file libtrac.c.

```
{

  /* Set timer... */
  SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
  LOG(2, "Extrapolate meteo data...");

  /* Loop over columns... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      int ip0;
      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 (int 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];
      }
    }
}
```

5.23.3.47 read_met_geopot()

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

Calculate geopotential heights.

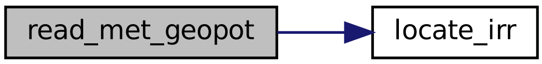
Definition at line 3392 of file libtrac.c.

```
{

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

```

```
  double logp[EP];

  int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;

  /* Set timer... */
  SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
  LOG(2, "Calculate geopotential heights...");

  /* Calculate log pressure... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < met->np; ip++)
    logp[ip] = log(met->p[ip]);

  /* Apply hydrostatic equation to calculate geopotential heights... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Get surface height and pressure... */
      double zs = met->zs[ix][iy];
      double lnps = log(met->ps[ix][iy]);

      /* Get temperature and water vapor vmr at the surface... */
      int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
      double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
                      met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
      double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
                        met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (zs +
                   ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
                         met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
      for (int ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] +
                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
                           met->h2o[ix][iy][ip - 1], logp[ip],
                           met->t[ix][iy][ip], met->h2o[ix][iy][ip]));

      /* Lower part of profile... */
      met->z[ix][iy][ip0]
        = (float) (zs +
                   ZDIFF(lnps, ts, h2os, logp[ip0],
                         met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
      for (int ip = ip0 - 1; ip >= 0; ip--)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip + 1] +
                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
                           met->h2o[ix][iy][ip + 1], logp[ip],
                           met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
    }

  /* Check control parameters... */
  if (dx == 0 || dy == 0)
    return;

  /* Default smoothing parameters... */
  if (dx < 0 || dy < 0) {
    if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
      dx = 3;
      dy = 2;
    } else {
      dx = 6;
      dy = 4;
    }
  }

  /* Calculate weights for smoothing... */
  float ws[dx + 1][dy + 1];
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix <= dx; ix++)
    for (int iy = 0; iy < dy; iy++)
      ws[ix][iy] = (1.0f - (float) ix / (float) dx)
        * (1.0f - (float) iy / (float) dy);

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        help[ip][ix][iy] = met->z[ix][iy][ip];

  /* Horizontal smoothing... */
#pragma omp parallel for default(shared) collapse(3)
  for (int ip = 0; ip < met->np; ip++)
```

```
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        float res = 0, wsum = 0;
        int iy0 = GSL_MAX(iy - dy + 1, 0);
        int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
        for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
          int ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (int iy2 = iy0; iy2 <= iy1; ++iy2)
            if (isfinite(help[ip][ix3][iy2])) {
              float w = ws[abs(ix - ix2)][abs(iy - iy2)];
              res += w * help[ip][ix3][iy2];
              wsum += w;
            }
        }
        if (wsum > 0)
          met->z[ix][iy][ip] = res / wsum;
        else
          met->z[ix][iy][ip] = GSL_NAN;
      }
}
```

Here is the call graph for this function:



5.23.3.48 read_met_grid()

```
void read_met_grid (
    char * filename,
    int ncid,
    ctl_t * ctl,
    met_t * met )
```

Read coordinates of meteo data.

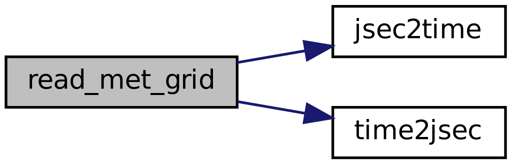
Definition at line 3512 of file libtrac.c.

```
                  {

  char levname[LEN], tstr[10];

  double rtime, r2;

  int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;

  size_t np;

  /* Set timer... */
  SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
  LOG(2, "Read meteo grid information...");

  /* MPTRAC meteo files... */
  if (ctl->clams_met_data == 0) {

    /* Get time from filename... */
    size_t len = strlen(filename);
    sprintf(tstr, "%.4s", &filename[len - 16]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[len - 11]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[len - 8]);
```

```
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[len - 5]);
    hour = atoi(tstr);
    time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

    /* Check time information from data file... */
    if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
      NC(nc_get_var_double(ncid, varid, &rtime));
      if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
        WARN("Time information in meteo file does not match filename!");
    } else
      WARN("Time information in meteo file is missing!");
  }

  /* CLaMS meteo files... */
  else {

    /* Read time from file... */
    NC_GET_DOUBLE("time", &rtime, 0);

    /* Get time from filename (considering the century)... */
    if (rtime < 0)
      sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
    else
      sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
    hour = atoi(tstr);
    time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
  }

  /* Check time... */
  if (year < 1900 || year > 2100 || mon < 1 || mon > 12
      || day < 1 || day > 31 || hour < 0 || hour > 23)
    ERRMSG("Cannot read time from filename!");
  jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
  LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
      met->time, year2, mon2, day2, hour2, min2);

  /* Get grid dimensions... */
  NC_INQ_DIM("lon", &met->nx, 2, EX);
  LOG(2, "Number of longitudes: %d", met->nx);

  NC_INQ_DIM("lat", &met->ny, 2, EY);
  LOG(2, "Number of latitudes: %d", met->ny);

  if (ctl->vert_coord_met == 0) {
    int dimid;
    sprintf(levname, "lev");
    if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR)
      sprintf(levname, "plev");
  } else
    sprintf(levname, "hybrid");
  NC_INQ_DIM(levname, &met->np, 1, EP);
  if (met->np == 1) {
    int dimid;
    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));
    met->np = (int) np;
  }
  LOG(2, "Number of levels: %d", met->np);
  if (met->np < 2 || met->np > EP)
    ERRMSG("Number of levels out of range!");

  /* Read longitudes and latitudes... */
  NC_GET_DOUBLE("lon", met->lon, 1);
  LOG(2, "Longitudes: %g, %g ... %g deg",
      met->lon[0], met->lon[1], met->lon[met->nx - 1]);
  NC_GET_DOUBLE("lat", met->lat, 1);
  LOG(2, "Latitudes: %g, %g ... %g deg",
      met->lat[0], met->lat[1], met->lat[met->ny - 1]);

  /* Read pressure levels... */
  if (ctl->met_np <= 0) {
    NC_GET_DOUBLE(levname, met->p, 1);
    for (int ip = 0; ip < met->np; ip++)
      met->p[ip] /= 100.;
    LOG(2, "Altitude levels: %g, %g ... %g km",
        Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
```

```
    LOG(2, "Pressure levels: %g, %g ... %g hPa",
        met->p[0], met->p[1], met->p[met->np - 1]);
  }
}
```

Here is the call graph for this function:



5.23.3.49 read_met_levels()

```
void read_met_levels (
    int ncid,
    ctl_t * ctl,
    met_t * met )
```

Read meteo data on vertical levels.

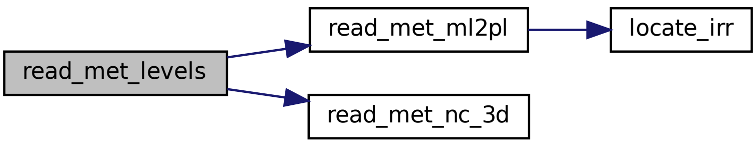
Definition at line 3634 of file libtrac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
  LOG(2, "Read level data...");

  /* MPTRAC meteo data... */
  if (ctl->clams_met_data == 0) {

    /* Read meteo data... */
    if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
      ERRMSG("Cannot read temperature!");
    if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
      ERRMSG("Cannot read zonal wind!");
    if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
      ERRMSG("Cannot read meridional wind!");
    if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
      WARN("Cannot read vertical velocity!");
    if (!read_met_nc_3d
        (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
      WARN("Cannot read specific humidity!");
    if (!read_met_nc_3d
        (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
      WARN("Cannot read ozone data!");
    if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
        WARN("Cannot read cloud liquid water content!");
      if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
        WARN("Cannot read cloud ice water content!");
    }
    if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d
          (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud rain water content!");
      if (!read_met_nc_3d
          (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud snow water content!");
    }
```

```
    if (ctl->met_relhum) {
      if (!read_met_nc_3d(ncid, "rh", "RH", ctl, met, met->h2o, 0.01f, 1))
        WARN("Cannot read relative humidity!");
#pragma omp parallel for default(shared) collapse(2)
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          for (int ip = 0; ip < met->np; ip++) {
            double pw = met->h2o[ix][iy][ip] * PSAT(met->t[ix][iy][ip]);
            met->h2o[ix][iy][ip] =
              (float) (pw / (met->p[ip] - (1.0 - EPS) * pw));
          }
    }

    /* Transfer from model levels to pressure levels... */
    if (ctl->met_np > 0) {

      /* Read pressure on model levels... */
      if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->pl, 0.01f, 1))
        ERRMSG("Cannot read pressure on model levels!");

      /* Vertical interpolation from model 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 new pressure levels... */
      met->np = ctl->met_np;
      for (int ip = 0; ip < met->np; ip++)
        met->p[ip] = ctl->met_p[ip];
    }

  }

  /* CLaMS meteo data... */
  else if (ctl->clams_met_data == 1) {

    /* Read meteorological data... */
    if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
      ERRMSG("Cannot read temperature!");
    if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
      ERRMSG("Cannot read zonal wind!");
    if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
      ERRMSG("Cannot read meridional wind!");
    if (!read_met_nc_3d(ncid, "W", "OMEGA", ctl, met, met->w, 0.01f, 1))
      WARN("Cannot read vertical velocity!");
    if (!read_met_nc_3d(ncid, "ZETA", "zeta", ctl, met, met->zeta, 1.0, 1))
      WARN("Cannot read ZETA in meteo data!");
    if (ctl->vert_vel == 1) {
      if (!read_met_nc_3d
          (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
           0.00001157407f, 1)) {
        if (!read_met_nc_3d
            (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
             0.00001157407f, 1)) {
          WARN("Cannot read vertical velocity!");
        }
      }
    }
    if (!read_met_nc_3d
        (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
      WARN("Cannot read specific humidity!");
    if (!read_met_nc_3d
        (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
      WARN("Cannot read ozone data!");
    if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
        WARN("Cannot read cloud liquid water content!");
      if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
        WARN("Cannot read cloud ice water content!");
    }
    if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
      if (!read_met_nc_3d
          (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud rain water content!");
      if (!read_met_nc_3d
          (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
           ctl->met_cloud == 2))
        WARN("Cannot read cloud snow water content!");
    }

    /* Transfer from model levels to pressure levels... */
```

```
    if (ctl->met_np > 0) {

      /* Read pressure on model levels... */
      if (!read_met_nc_3d(ncid, "pl", "PRESS", ctl, met, met->pl, 1.0, 1))
        ERRMSG("Cannot read pressure on model levels!");

      /* Vertical interpolation from model 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);
      if (ctl->vert_vel == 1) {
        read_met_ml2pl(ctl, met, met->zeta);
        read_met_ml2pl(ctl, met, met->zeta_dot);
      }

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

      /* Create a pressure field... */
      for (int i = 0; i < met->nx; i++)
        for (int j = 0; j < met->ny; j++)
          for (int k = 0; k < met->np; k++) {
            met->patp[i][j][k] = (float) met->p[k];
          }
    }
  } else
    ERRMSG("Meteo data format unknown!");

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

Here is the call graph for this function:



5.23.3.50 read_met_ml2pl()

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

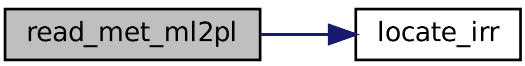
Convert meteo data from model levels to pressure levels.

Definition at line 3807 of file libtrac.c.

```
                              {

  double aux[EP], p[EP];

  /* Set timer... */
  SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
  LOG(2, "Interpolate meteo data to pressure levels...");
```

```

  /* Loop over columns... */
#pragma omp parallel for default(shared) private(aux,p) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

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

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

Here is the call graph for this function:



5.23.3.51 read_met_nc_2d()

```
int read_met_nc_2d (
    int ncid,
    char * varname,
    char * varname2,
    ctl_t * ctl,
    met_t * met,
    float dest[EX][EY],
    float scl,
    int init )
```

Read and convert 2D variable from meteo data file.

Definition at line 3848 of file libtrac.c.

```
            {

  char varsel[LEN];

  float offset, scalfac;

  int varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
      WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
      return 0;
    } else {
```

```
      sprintf(varsel, "%s", varname2);
  } else
    sprintf(varsel, "%s", varname);

  /* Read packed data... */
  if (ctl->met_nc_scale
      && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
      && nc_get_att_float(ncid, varid, "scale_factor",
                          &scalfac) == NC_NOERR) {

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

    /* Read fill value and missing value... */
    short fillval, missval;
    if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 2-D variable: %s"
        " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
        varsel, fillval, missval, scalfac, offset);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        if (init)
          dest[ix][iy] = 0;
        short aux = help[ARRAY_2D(iy, ix, met->nx)];
        if ((fillval == 0 || aux != fillval)
            && (missval == 0 || aux != missval)
            && fabsf(aux * scalfac + offset) < 1e14f)
          dest[ix][iy] += scl * (aux * scalfac + offset);
        else
          dest[ix][iy] = GSL_NAN;
      }

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

  /* Unpacked data... */
  else {

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

    /* Read fill value and missing value... */
    float fillval, missval;
    if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
        varsel, fillval, missval);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {
        if (init)
          dest[ix][iy] = 0;
        float aux = help[ARRAY_2D(iy, ix, met->nx)];
        if ((fillval == 0 || aux != fillval)
            && (missval == 0 || aux != missval)
            && fabsf(aux) < 1e14f)
          dest[ix][iy] += scl * aux;
        else
          dest[ix][iy] = GSL_NAN;
      }

    /* Free... */
```

```
    free(help);
  }

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

5.23.3.52 read_met_nc_3d()

```
int read_met_nc_3d (
        int ncid,
        char * varname,
        char * varname2,
        ctl_t * ctl,
        met_t * met,
        float dest[EX][EY][EP],
        float scl,
        int init )
```

Read and convert 3D variable from meteo data file.

Definition at line 3966 of file libtrac.c.

```
              {

  char varsel[LEN];

  float offset, scalfac;

  int varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
      WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
      return 0;
    } else {
      sprintf(varsel, "%s", varname2);
  } else
    sprintf(varsel, "%s", varname);

  /* Read packed data... */
  if (ctl->met_nc_scale
      && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
      && nc_get_att_float(ncid, varid, "scale_factor",
                          &scalfac) == NC_NOERR) {

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

    /* Read fill value and missing value... */
    short fillval, missval;
    if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 3-D variable: %s "
        "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
        varsel, fillval, missval, scalfac, offset);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        for (int ip = 0; ip < met->np; ip++) {
          if (init)
            dest[ix][iy][ip] = 0;
          short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
          if ((fillval == 0 || aux != fillval)
              && (missval == 0 || aux != missval)
              && fabsf(aux * scalfac + offset) < 1e14f)
```

```
            dest[ix][iy][ip] += scl * (aux * scalfac + offset);
          else
            dest[ix][iy][ip] = GSL_NAN;
        }

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

  /* Unpacked data... */
  else {

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

    /* Read fill value and missing value... */
    float fillval, missval;
    if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
      fillval = 0;
    if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
      missval = 0;

    /* Write info... */
    LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
        varsel, fillval, missval);

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

    /* Copy and check data... */
#pragma omp parallel for default(shared) num_threads(12)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        for (int ip = 0; ip < met->np; ip++) {
          if (init)
            dest[ix][iy][ip] = 0;
          float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
          if ((fillval == 0 || aux != fillval)
              && (missval == 0 || aux != missval)
              && fabsf(aux) < 1e14f)
            dest[ix][iy][ip] += scl * aux;
          else
            dest[ix][iy][ip] = GSL_NAN;
        }

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

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

5.23.3.53 read_met_pbl() `void read_met_pbl ( met_t * met )`

Calculate pressure of the boundary layer.

Definition at line 4086 of file libtrac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
  LOG(2, "Calculate planetary boundary layer...");

  /* Parameters used to estimate the height of the PBL
     (e.g., Vogelezang and Holtslag, 1996; Seidel et al., 2012)... */
  const double rib_crit = 0.25, dz = 0.05, umin = 5.0;

  /* Loop over grid points... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {

      /* Set bottom level of PBL... */
      double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);

```

```
      /* Find lowest level near the bottom... */
      int ip;
      for (ip = 1; ip < met->np; ip++)
        if (met->p[ip] < pbl_bot)
          break;

      /* Get near surface data... */
      double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
                      met->p[ip], met->z[ix][iy][ip], pbl_bot);
      double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
                      met->p[ip], met->t[ix][iy][ip], pbl_bot);
      double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
                      met->p[ip], met->u[ix][iy][ip], pbl_bot);
      double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
                      met->p[ip], met->v[ix][iy][ip], pbl_bot);
      double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
                        met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
      double tvs = THETAVIRT(pbl_bot, ts, h2os);

      /* Init... */
      double rib_old = 0;

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

        /* Get squared horizontal wind speed... */
        double vh2
          = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
        vh2 = GSL_MAX(vh2, SQR(umin));

        /* Calculate bulk Richardson number... */
        double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
          * (THETAVIRT(met->p[ip], met->t[ix][iy][ip],
                       met->h2o[ix][iy][ip]) - tvs) / vh2;

        /* Check for critical value... */
        if (rib >= rib_crit) {
          met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
                                          rib, met->p[ip], rib_crit));
          if (met->pbl[ix][iy] > pbl_bot)
            met->pbl[ix][iy] = (float) pbl_bot;
          break;
        }

        /* Save Richardson number... */
        rib_old = rib;
      }
    }
}
```

5.23.3.54 read_met_periodic() `void read_met_periodic ( met_t * met )`

Create meteo data with periodic boundary conditions.

Definition at line 4157 of file libtrac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
  LOG(2, "Apply periodic boundary conditions...");

  /* 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];
```

```
    met->ts[met->nx - 1][iy] = met->ts[0][iy];
    met->us[met->nx - 1][iy] = met->us[0][iy];
    met->vs[met->nx - 1][iy] = met->vs[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];
    }
  }
}
```

5.23.3.55 read_met_pv() void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 4199 of file libtrac.c.

```
                 {

  double pows[EP];

  /* Set timer... */
  SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
  LOG(2, "Calculate potential vorticity...");

  /* Set powers... */
#pragma omp parallel for default(shared)
  for (int 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)
  for (int ix = 0; ix < met->nx; ix++) {

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

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

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

      /* Set auxiliary variables... */
      double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
      double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
      double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
      double c0 = cos(met->lat[iy0] / 180. * M_PI);
      double c1 = cos(met->lat[iy1] / 180. * M_PI);
      double cr = cos(latr / 180. * M_PI);
      double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);

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

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

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

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

        /* Get gradients in pressure... */
        double dtdp, dudp, dvdp;
```

```
        double dp0 = 100. * (met->p[ip] - met->p[ip0]);
        double dp1 = 100. * (met->p[ip1] - met->p[ip]);
        if (ip != ip0 && ip != ip1) {
          double 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 {
          double 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)
  for (int ix = 0; ix < met->nx; ix++)
    for (int 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];
    }
}
```

5.23.3.56 read_met_sample()

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

Downsampling of meteo data.

Definition at line 4305 of file libtrac.c.

```
                {

  met_t *help;

  /* 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;

  /* Set timer... */
  SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
  LOG(2, "Downsampling of meteo data...");

  /* 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 (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
    for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
      for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
        help->ps[ix][iy] = 0;
        help->zs[ix][iy] = 0;
        help->ts[ix][iy] = 0;
        help->us[ix][iy] = 0;
        help->vs[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;
        float wsum = 0;
        for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
             ix2++) {
          int ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
            for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
              float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
                * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
                * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->zs[ix][iy] += w * met->zs[ix3][iy2];
              help->ts[ix][iy] += w * met->ts[ix3][iy2];
              help->us[ix][iy] += w * met->us[ix3][iy2];
              help->vs[ix][iy] += w * met->vs[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->ts[ix][iy] /= wsum;
        help->us[ix][iy] /= wsum;
        help->vs[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 (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (int 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->ts[met->nx][met->ny] = help->ts[ix][iy];
      met->us[met->nx][met->ny] = help->us[ix][iy];
      met->vs[met->nx][met->ny] = help->vs[ix][iy];
      met->np = 0;
      for (int 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);
}
```

5.23.3.57 read_met_surface()

```
void read_met_surface (
    int ncid,
    met_t * met,
    ctl_t * ctl )
```

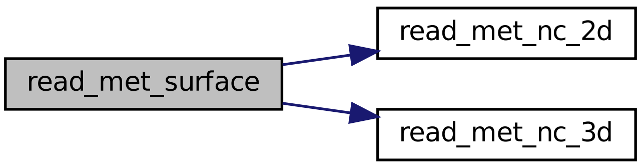
Read surface data.

Definition at line 4433 of file libtrac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
  LOG(2, "Read surface data...");

  /* MPTRAC meteo data... */
  if (ctl->clams_met_data == 0) {

    /* Read surface pressure... */
    if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
      if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
        if (!read_met_nc_2d(ncid, "sp", "SP", ctl, met, met->ps, 0.01f, 1)) {
          WARN("Cannot not read surface pressure data (use lowest level)!");
          for (int ix = 0; ix < met->nx; ix++)
            for (int iy = 0; iy < met->ny; iy++)
              met->ps[ix][iy] = (float) met->p[0];
        }
      }
    } else
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);

    /* Read geopotential height at the surface... */
    if (!read_met_nc_2d
        (ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
      if (!read_met_nc_2d
          (ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
        WARN("Cannot read surface geopotential height!");

    /* Read temperature at the surface... */
    if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
      WARN("Cannot read surface temperature!");

    /* Read zonal wind at the surface... */
    if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
      WARN("Cannot read surface zonal wind!");

    /* Read meridional wind at the surface... */
    if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
      WARN("Cannot read surface meridional wind!");
  }

  /* CLaMS meteo data... */
  else {

    /* Read surface pressure... */
    if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
      WARN("Cannot not read surface pressure data (use lowest level)!");
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->ps[ix][iy] = (float) met->p[0];
    }

    /* Read geopotential height at the surface
       (use lowermost level of 3-D data field)... */
    float *help;
```

```
    ALLOC(help, float,
          EX * EY * EP);
    memcpy(help, met->pl, sizeof(met->pl));
    if (!read_met_nc_3d
        (ncid, "gph", "GPH", ctl, met, met->pl, (float) (1e-3 / G0), 1)) {
      ERRMSG("Cannot read geopotential height!");
    } else
      for (int ix = 0; ix < met->nx; ix++)
        for (int iy = 0; iy < met->ny; iy++)
          met->zs[ix][iy] = met->pl[ix][iy][0];
    memcpy(met->pl, help, sizeof(met->pl));
    free(help);

    /* Read temperature at the surface... */
    if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
      WARN("Cannot read surface temperature!");

    /* Read zonal wind at the surface... */
    if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
      WARN("Cannot read surface zonal wind!");

    /* Read meridional wind at the surface... */
    if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
      WARN("Cannot read surface meridional wind!");
  }
}
```

Here is the call graph for this function:



5.23.3.58 read_met_tropo()

```
void read_met_tropo (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met )
```

Calculate tropopause data.

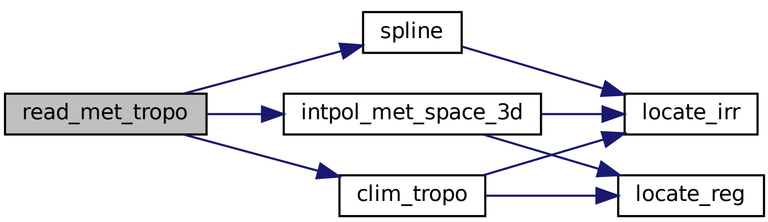
Definition at line 4523 of file libtrac.c.

```
                 {

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

  /* Set timer... */
  SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
  LOG(2, "Calculate tropopause...");

  /* Get altitude and pressure profiles... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < met->np; iz++)
    z[iz] = Z(met->p[iz]);
#pragma omp parallel for default(shared)
  for (int 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)
#pragma omp parallel for default(shared) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int 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) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++)
        met->pt[ix][iy] = (float) clim_tropo(clim, 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(t,t2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        for (int iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);

        /* Find minimum... */
        int 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(t,t2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate temperature profile... */
        int iz;
        for (iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);

        /* Find 1st tropopause... */
        met->pt[ix][iy] = GSL_NAN;
        for (iz = 0; iz <= 170; iz++) {
          int found = 1;
          for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
            if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
                ctl->met_tropo_lapse) {
              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++) {
            int found = 1;
            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
                  ctl->met_tropo_lapse_sep) {
                found = 0;
                break;
              }
            if (found)
              break;
          }
          for (; iz <= 170; iz++) {
            int found = 1;
            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
```

```
                  ctl->met_tropo_lapse) {
                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(pv,pv2,th,th2) collapse(2)
    for (int ix = 0; ix < met->nx; ix++)
      for (int iy = 0; iy < met->ny; iy++) {

        /* Interpolate potential vorticity profile... */
        for (int iz = 0; iz < met->np; iz++)
          pv[iz] = met->pv[ix][iy][iz];
        spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);

        /* Interpolate potential temperature profile... */
        for (int 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, ctl->met_tropo_spline);

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

  else
    ERRMSG("Cannot calculate tropopause!");

  /* Interpolate temperature, geopotential height, and water vapor vmr... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++) {
      double h2ot, tt, zt;
      INTPOL_INIT;
      intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &tt, ci, cw, 1);
      intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &zt, ci, cw, 0);
      intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
                          met->lat[iy], &h2ot, ci, cw, 0);
      met->tt[ix][iy] = (float) tt;
      met->zt[ix][iy] = (float) zt;
      met->h2ot[ix][iy] = (float) h2ot;
    }
}
```

Here is the call graph for this function:



5.23.3.59 read_obs() void read_obs (

```
char * filename,
double * rt,
double * rz,
double * rlon,
double * rlat,
double * robs,
int * nobs )
```

Read observation data.

Definition at line 4698 of file libtrac.c.

```
               {

  FILE *in;

  char line[LEN];

  /* Open observation data file... */
  LOG(1, "Read observation data: %s", filename);
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read observations... */
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
               &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
      if ((++(*nobs)) >= NOBS)
        ERRMSG("Too many observations!");

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

  /* Check time... */
  for (int i = 1; i < *nobs; i++)
    if (rt[i] < rt[i - 1])
      ERRMSG("Time must be ascending!");

  /* Write info... */
  int n = *nobs;
  double mini, maxi;
  LOG(2, "Number of observations: %d", *nobs);
  gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
  LOG(2, "Observation range: %g ... %g", mini, maxi);
}
```

5.23.3.60 scan_ctl() 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 4749 of file libtrac.c.

```
                {

  FILE *in = NULL;

  char fullname1[LEN], fullname2[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) {
    char dummy[LEN], line[LEN], rvarname[LEN];
    while (fgets(line, LEN, in)) {
      if (sscanf(line, "%4999s %4999s %4999s", 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
      ERRMSG("Missing variable %s!\n", fullname1);
  }

  /* Write info... */
  LOG(1, "%s = %s", fullname1, rval);

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

5.23.3.61 sedi()

```
double sedi (
    double p,
    double T,
    double rp,
    double rhop )
```

Calculate sedimentation velocity.

Definition at line 4821 of file libtrac.c.

```
                {

  /* Convert particle radius from microns to m... */
  rp *= 1e-6;

```

```
  /* Density of dry air [kg / m^3]... */
  double rho = RHO(p, T);

  /* Dynamic viscosity of air [kg / (m s)]... */
  double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Thermal velocity of an air molecule [m / s]... */
  double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));

  /* Mean free path of an air molecule [m]... */
  double lambda = 2. * eta / (rho * v);

  /* Knudsen number for air (dimensionless)... */
  double K = lambda / rp;

  /* Cunningham slip-flow correction (dimensionless)... */
  double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));

  /* Sedimentation velocity [m / s]... */
  return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
}
```

5.23.3.62 spline()

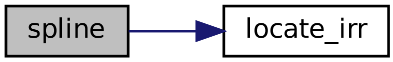
```
void spline (
        double * x,
        double * y,
        int n,
        double * x2,
        double * y2,
        int n2,
        int method )
```

Spline interpolation.

Definition at line 4854 of file libtrac.c.

```
                {

  /* Cubic spline interpolation... */
  if (method == 1) {

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

    /* Interpolate 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);
  }

  /* Linear interpolation... */
  else {
    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 {
        int idx = locate_irr(x, n, x2[i]);
        y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
      }
  }
}
```

Here is the call graph for this function:



5.23.3.63 stddev()

```
float stddev (
    float * data,
    int n )
```

Calculate standard deviation.

Definition at line 4903 of file libtrac.c.

```
          {

  if (n <= 0)
    return 0;

  float mean = 0, var = 0;

  for (int i = 0; i < n; ++i) {
    mean += data[i];
    var += SQR(data[i]);
  }

  var = var / (float) n - SQR(mean / (float) n);

  return (var > 0 ? sqrtf(var) : 0);
}
```

5.23.3.64 sza()

```
double sza (
    double sec,
    double lon,
    double lat )
```

Calculate solar zenith angle.

Definition at line 4924 of file libtrac.c.

```
              {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

```

```
  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [rad]... */
  return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
}
```

5.23.3.65 time2jsec()

```
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 4966 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;
}
```

5.23.3.66 timer()

```
void timer (
    const char * name,
    const char * group,
    int output )
```

Measure wall-clock time.

Definition at line 4997 of file libtrac.c.

```
              {

  static char names[NTIMER][100], groups[NTIMER][100];

  static double rt_name[NTIMER], rt_group[NTIMER],
```

```
  rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;

static int iname = -1, igroup = -1, nname, ngroup, ct_name[NTIMER];

/* Get time... */
t1 = omp_get_wtime();
dt = t1 - t0;

/* Add elapsed time to current timers... */
if (iname >= 0) {
  rt_name[iname] += dt;
  rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
  rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
  ct_name[iname]++;
}
if (igroup >= 0)
  rt_group[igroup] += t1 - t0;

/* Report timers... */
if (output) {
  for (int i = 0; i < nname; i++)
    LOG(1, "TIMER_%s = %.3f s    (min= %g s, mean= %g s,"
        " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
        rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
  for (int i = 0; i < ngroup; i++)
    LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
  double total = 0.0;
  for (int i = 0; i < nname; i++)
    total += rt_name[i];
  LOG(1, "TIMER_TOTAL = %.3f s", total);
}

/* Identify IDs of next timer... */
for (iname = 0; iname < nname; iname++)
  if (strcasecmp(name, names[iname]) == 0)
    break;
for (igroup = 0; igroup < ngroup; igroup++)
  if (strcasecmp(group, groups[igroup]) == 0)
    break;

/* Check whether this is a new timer... */
if (iname >= nname) {
  sprintf(names[iname], "%s", name);
  if ((++nname) > NTIMER)
    ERRMSG("Too many timers!");
}

/* Check whether this is a new group... */
if (igroup >= ngroup) {
  sprintf(groups[igroup], "%s", group);
  if ((++ngroup) > NTIMER)
    ERRMSG("Too many groups!");
}

/* Save starting time... */
t0 = t1;
}
```

5.23.3.67 tropo_weight()

```
double tropo_weight (
    clim_t * clim,
    double t,
    double lat,
    double p )
```

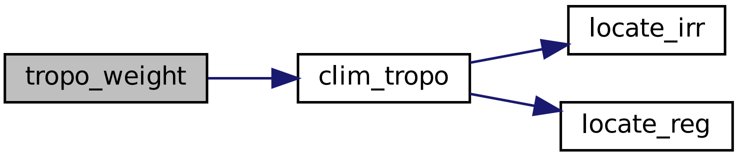
Get weighting factor based on tropopause distance.

Definition at line 5065 of file libtrac.c.

```
              {

/* Get tropopause pressure... */
double pt = clim_tropo(clim, t, lat);

/* Get pressure range... */
double p1 = pt * 0.866877899;
double p0 = pt / 0.866877899;

/* Get weighting factor... */
```

```
  if (p > p0)
    return 1;
  else if (p < p1)
    return 0;
  else
    return LIN(p0, 1.0, p1, 0.0, p);
}
```

Here is the call graph for this function:



5.23.3.68 write_atm()

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

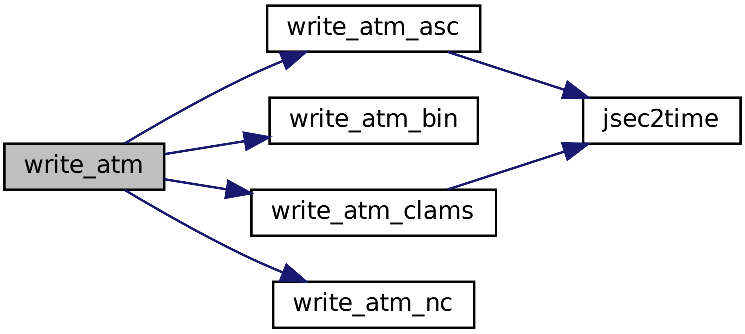
Write atmospheric data.

Definition at line 5089 of file libtrac.c.

```
            {

  /* Set timer... */
  SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write atmospheric data: %s", filename);

  /* Write ASCII data... */
  if (ctl->atm_type == 0)
    write_atm_asc(filename, ctl, atm, t);

  /* Write binary data... */
  else if (ctl->atm_type == 1)
    write_atm_bin(filename, ctl, atm);

  /* Write netCDF data... */
  else if (ctl->atm_type == 2)
    write_atm_nc(filename, ctl, atm);

  /* Write CLaMS data... */
  else if (ctl->atm_type == 3)
    write_atm_clams(ctl, atm, t);

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of particles: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
```

```
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  for (int iq = 0; iq < ctl->nq; iq++) {
    char msg[LEN];
    sprintf(msg, "Quantity %s range: %s ... %s %s",
            ctl->qnt_name[iq], ctl->qnt_format[iq],
            ctl->qnt_format[iq], ctl->qnt_unit[iq]);
    gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
    LOG(2, msg, mini, maxi);
  }
}
```

Here is the call graph for this function:



5.23.3.69 write_atm_asc()

```
void write_atm_asc (
        const char * filename,
        ctl_t * ctl,
        atm_t * atm,
        double t )
```

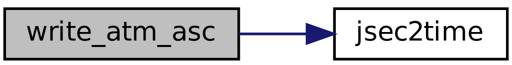
Write atmospheric data in ASCII format.

Definition at line 5145 of file libtrac.c.

```
             {

  FILE *out;

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

  /* 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... */
    double r;
    int year, mon, day, hour, min, sec;
    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... */
    FILE *in;
    if (!(in = fopen(ctl->atm_gpfile, "r")))
      ERRMSG("Cannot open file!");
    char line[LEN];
    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 (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");

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

    /* Check time... */
    if (ctl->atm_filter == 2 && (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 (int iq = 0; iq < ctl->nq; iq++) {
      fprintf(out, " ");
      if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
        fprintf(out, ctl->qnt_format[iq], GSL_NAN);
      else
        fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
    }
    fprintf(out, "\n");
  }

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

Here is the call graph for this function:



5.23.3.70 write_atm_bin()

```
void write_atm_bin (
    const char * filename,
    ctl_t * ctl,
    atm_t * atm )
```

Write atmospheric data in binary format.

Definition at line 5227 of file libtrac.c.

```
                    {

  FILE *out;

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

  /* Write version of binary data... */
  int version = 100;
  FWRITE(&version, int,
         1,
         out);

  /* 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 (int iq = 0; iq < ctl->nq; iq++)
    FWRITE(atm->q[iq], double,
             (size_t) atm->np,
           out);

  /* Write final flag... */
  int final = 999;
  FWRITE(&final, int,
         1,
         out);

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

5.23.3.71 write_atm_clams()

```
void write_atm_clams (
        ctl_t * ctl,
        atm_t * atm,
        double t )
```

Write atmospheric data in CLaMS format.

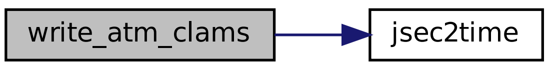
Definition at line 5277 of file libtrac.c.

```
                {

  /* Global Counter... */
  static size_t out_cnt = 0;

  char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";

  double r, r_start, r_stop;

  int year, mon, day, hour, min, sec;
  int year_start, mon_start, day_start, hour_start, min_start, sec_start;
  int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
  int ncid, varid, tid, pid, cid, zid, dim_ids[2];

  /* time, nparc */
  size_t start[2], count[2];

  /* Determine start and stop times of calculation... */
  jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
  jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
            &min_start, &sec_start, &r_start);
  jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
            &min_stop, &sec_stop, &r_stop);

```

```
  /* Set filename... */
  sprintf(filename_out,
          "./traj_fix_3d_%02d%02d%02d%02d_%02d%02d%02d%02d.nc",
          year_start % 100, mon_start, day_start, hour_start,
          year_stop % 100, mon_stop, day_stop, hour_stop);
  printf("Write traj file: %s\n", filename_out);

  /* Define hyperslap for the traj_file... */
  start[0] = out_cnt;
  start[1] = 0;
  count[0] = 1;
  count[1] = (size_t) atm->np;

  /* Create the file at the first timestep... */
  if (out_cnt == 0) {

    /* Create file... */
    nc_create(filename_out, NC_CLOBBER, &ncid);

    /* Define dimensions... */
    NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
    NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
    NC(nc_def_dim(ncid, "TMDT", 7, &cid));
    dim_ids[0] = tid;
    dim_ids[1] = pid;

    /* Define variables and their attributes... */
    NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
               "seconds since 2000-01-01 00:00:00 UTC");
    NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
    NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
    NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
    NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);

    /* Define global attributes... */
    NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
    NC_PUT_ATT_GLOBAL("model", "MPTRAC");

    /* End definitions... */
    NC(nc_enddef(ncid));
    NC(nc_close(ncid));
  }

  /* Increment global counter to change hyperslap... */
  out_cnt++;

  /* Open file... */
  NC(nc_open(filename_out, NC_WRITE, &ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", atm->time, 1);
  NC_PUT_DOUBLE("LAT", atm->lat, 1);
  NC_PUT_DOUBLE("LON", atm->lon, 1);
  NC_PUT_DOUBLE("PRESS", atm->p, 1);
  if (ctl->vert_coord_ap == 1) {
    NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
  } else if (ctl->qnt_zeta >= 0) {
    NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
  }
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);

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

  /* At the last time step create the init_fix_YYYYMMDDHH file... */
  if ((year == year_stop) && (mon == mon_stop)
      && (day == day_stop) && (hour == hour_stop)) {

    /* Set filename... */
    char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
    sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
            year_stop % 100, mon_stop, day_stop, hour_stop);
    printf("Write init file: %s\n", filename_init);

    /* Create file... */
    nc_create(filename_init, NC_CLOBBER, &ncid);

    /* Define dimensions... */
    NC(nc_def_dim(ncid, "time", 1, &tid));
    NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
    dim_ids[0] = tid;
    dim_ids[1] = pid;

```

```
    /* Define variables and their attributes... */
    NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
               "seconds since 2000-01-01 00:00:00 UTC");
    NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
    NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
    NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
    NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
    NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
    NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);

    /* Define global attributes... */
    NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
    NC_PUT_ATT_GLOBAL("model", "MPTRAC");

    /* End definitions... */
    NC(nc_enddef(ncid));

    /* Write data... */
    NC_PUT_DOUBLE("time", atm->time, 0);
    NC_PUT_DOUBLE("LAT", atm->lat, 0);
    NC_PUT_DOUBLE("LON", atm->lon, 0);
    NC_PUT_DOUBLE("PRESS", atm->p, 0);
    NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
    for (int iq = 0; iq < ctl->nq; iq++)
      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

Here is the call graph for this function:



5.23.3.72 write_atm_nc()

```
void write_atm_nc (
    const char * filename,
    ctl_t * ctl,
    atm_t * atm )
```

Write atmospheric data in netCDF format.

Definition at line 5427 of file libtrac.c.

```
               {

  int ncid, obsid, varid;

  size_t start[2], count[2];

  /* Create file... */
  nc_create(filename, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
```

```
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
               ctl->qnt_longname[iq], ctl->qnt_unit[iq]);

  /* Define global attributes... */
  NC_PUT_ATT_GLOBAL("featureType", "point");

  /* End definitions... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", atm->time, 0);
  NC_PUT_DOUBLE("press", atm->p, 0);
  NC_PUT_DOUBLE("lon", atm->lon, 0);
  NC_PUT_DOUBLE("lat", atm->lat, 0);
  for (int iq = 0; iq < ctl->nq; iq++)
    NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);

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

5.23.3.73 write_csi()

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

Write CSI data.

Definition at line 5472 of file libtrac.c.

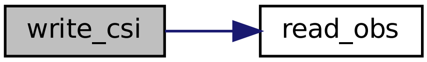
```
                {

  static FILE *out;

  static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
    dlon, dlat, dz, x[NCSI], y[NCSI];

  static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;

  /* Set timer... */
  SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);

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

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

    /* Allocate... */
    ALLOC(area, double,
          ctl->csi_ny);
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create new file... */
    LOG(1, "Write CSI data: %s", 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 (ratio of forecasts and observations) [%%]\n"
            "# $8 = probability of detection (POD) [%%]\n"
            "# $9 = false alarm rate (FAR) [%%]\n"
            "# $10 = critical success index (CSI) [%%]\n");
    fprintf(out,
            "# $11 = hits associated with random chance\n"
            "# $12 = equitable threat score (ETS) [%%]\n"
            "# $13 = Pearson linear correlation coefficient\n"
            "# $14 = Spearman rank-order correlation coefficient\n"
            "# $15 = column density mean error (F - O) [kg/m^2]\n"
            "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
            "# $17 = column density mean absolute error [kg/m^2]\n"
            "# $18 = number of data points\n\n");

    /* Set grid box size... */
    dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
    dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
    dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;

    /* Set horizontal coordinates... */
    for (iy = 0; iy < ctl->csi_ny; iy++) {
      double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
      area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
    }
  }

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

  /* Allocate... */
  ALLOC(modmean, double,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
  ALLOC(obsmean, double,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
  ALLOC(obscount, int,
        ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Check observation data... */
    if (!isfinite(robs[i]))
      continue;

    /* Calculate indices... */
    ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
    iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
    iz = (int) ((rz[i] - ctl->csi_z0) / dz);

    /* 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... */
    int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
    obsmean[idx] += robs[i];
    obscount[idx]++;
  }

  /* Analyze model data... */
  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) / dlon);
    iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
    iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);

    /* 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... */
    int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
    modmean[idx] += atm->q[ctl->qnt_m][ip];
  }

  /* Analyze all grid cells... */
  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... */
        int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
        if (obscount[idx] > 0)
          obsmean[idx] /= obscount[idx];

        /* Calculate column density... */
        if (modmean[idx] > 0)
          modmean[idx] /= (1e6 * area[iy]);

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

        /* Save data for other verification statistics... */
        if (obscount[idx] > 0
            && (obsmean[idx] >= ctl->csi_obsmin
                || modmean[idx] >= ctl->csi_modmin)) {
          x[n] = modmean[idx];
          y[n] = obsmean[idx];
          if ((++n) > NCSI)
            ERRMSG("Too many data points to calculate statistics!");
        }
      }

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

    /* Calculate verification statistics
       (https://www.cawcr.gov.au/projects/verification/) ... */
    static double work[2 * NCSI];
    int n_obs = cx + cy;
    int n_for = cx + cz;
    double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
    double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
    double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
    double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
    double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
    double ets = (cx + cy + cz - cx_rd > 0) ?
      (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
    double rho_p =
      (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
    double rho_s =
      (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
    for (int i = 0; i < n; i++)
      work[i] = x[i] - y[i];
    double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
    double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
                                                         0.0) : GSL_NAN;
    double absdev =
      (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;

    /* Write... */
    fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %g %g %d\n",
            t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
            rho_p, rho_s, mean, rmse, absdev, n);

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

  /* Free... */
  free(modmean);
  free(obsmean);
  free(obscount);
```

```
  /* Finalize... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(area);
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}
```

Here is the call graph for this function:



5.23.3.74 write_ens()

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

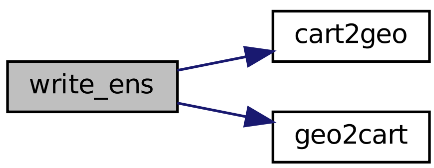
Write ensemble data.

Definition at line 5712 of file libtrac.c.

```
                 {

  static FILE *out;

  static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
    x[3], zm[NENS];

  static int n[NENS];

  /* Set timer... */
  SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);

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

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

  /* Init... */
  for (int i = 0; i < NENS; i++) {
    for (int iq = 0; iq < ctl->nq; iq++)
      qm[iq][i] = qs[iq][i] = 0;
    xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
    n[i] = 0;
  }

  /* 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 ensemble ID... */
    if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
      ERRMSG("Ensemble ID is out of range!");

    /* Get means... */
    geo2cart(0, atm->lon[ip], atm->lat[ip], x);
    for (int iq = 0; iq < ctl->nq; iq++) {
      qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
      qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
    }
    xm[ctl->qnt_ens][0] += x[0];
    xm[ctl->qnt_ens][1] += x[1];
    xm[ctl->qnt_ens][2] += x[2];
    zm[ctl->qnt_ens] += Z(atm->p[ip]);
    n[ctl->qnt_ens]++;
  }

  /* Create file... */
  LOG(1, "Write ensemble data: %s", 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 (int iq = 0; iq < ctl->nq; iq++)
    fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
            ctl->qnt_name[iq], ctl->qnt_unit[iq]);
  for (int 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);

  /* Write data... */
  for (int i = 0; i < NENS; i++)
    if (n[i] > 0) {
      cart2geo(xm[i], &dummy, &lon, &lat);
      fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
      for (int iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
      }
      for (int iq = 0; iq < ctl->nq; iq++) {
        fprintf(out, " ");
        double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
        fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
      }
      fprintf(out, " %d\n", n[i]);
    }

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

Here is the call graph for this function:



5.23.3.75 write_grid()

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

Write gridded data.

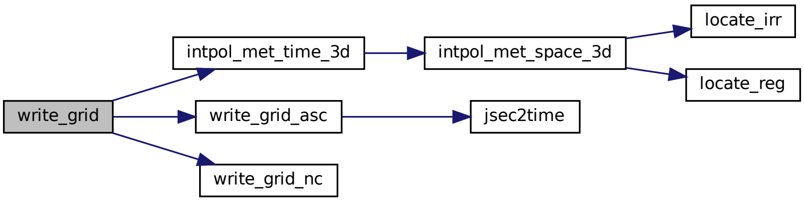
Definition at line 5809 of file libtrac.c.

```
                    {

  double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;

  int *ixs, *iys, *izs, *np;

  /* Set timer... */
  SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write grid data: %s", filename);

  /* Allocate... */
  ALLOC(cd, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(mass, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(vmr_expl, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(vmr_impl, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(z, double,
        ctl->grid_nz);
  ALLOC(lon, double,
        ctl->grid_nx);
  ALLOC(lat, double,
        ctl->grid_ny);
  ALLOC(area, double,
        ctl->grid_ny);
  ALLOC(press, double,
        ctl->grid_nz);
  ALLOC(np, int,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(ixs, int,
        atm->np);
  ALLOC(iys, int,
        atm->np);
  ALLOC(izs, int,
        atm->np);

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

  /* Set vertical coordinates... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < ctl->grid_nz; iz++) {
    z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
    press[iz] = P(z[iz]);
  }

  /* Set horizontal coordinates... */
  for (int ix = 0; ix < ctl->grid_nx; ix++)
    lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
#pragma omp parallel for default(shared)
  for (int iy = 0; iy < ctl->grid_ny; iy++) {
    lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
    area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
      * cos(lat[iy] * M_PI / 180.);
  }

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

  /* Get grid box indices... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++) {
    ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
    iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
```

```
    izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
    if (atm->time[ip] < t0 || atm->time[ip] > t1
        || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
        || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
        || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
      izs[ip] = -1;
  }

  /* Average data... */
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0) {
      int idx =
        ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
      np[idx]++;
      if (ctl->qnt_m >= 0)
        mass[idx] += atm->q[ctl->qnt_m][ip];
      if (ctl->qnt_vmr >= 0)
        vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
    }

  /* Calculate column density and vmr... */
#pragma omp parallel for default(shared)
  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++) {

        /* Get grid index... */
        int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);

        /* Calculate column density... */
        cd[idx] = GSL_NAN;
        if (ctl->qnt_m >= 0)
          cd[idx] = mass[idx] / (1e6 * area[iy]);

        /* Calculate volume mixing ratio (implicit)... */
        vmr_impl[idx] = GSL_NAN;
        if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
          vmr_impl[idx] = 0;
          if (mass[idx] > 0) {

            /* Get temperature... */
            double temp;
            INTPOL_INIT;
            intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
                               lon[ix], lat[iy], &temp, ci, cw, 1);

            /* Calculate volume mixing ratio... */
            vmr_impl[idx] = MA / ctl->molmass * mass[idx]
              / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
          }
        }

        /* Calculate volume mixing ratio (explicit)... */
        if (ctl->qnt_vmr >= 0 && np[idx] > 0)
          vmr_expl[idx] /= np[idx];
        else
          vmr_expl[idx] = GSL_NAN;
      }

  /* Write ASCII data... */
  if (ctl->grid_type == 0)
    write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
                   t, z, lon, lat, area, dz, np);

  /* Write netCDF data... */
  else if (ctl->grid_type == 1)
    write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
                  t, z, lon, lat, area, dz, np);

  /* Error message... */
  else
    ERRMSG("Grid data format GRID_TYPE unknown!");

  /* Free... */
  free(cd);
  free(mass);
  free(vmr_expl);
  free(vmr_impl);
  free(z);
  free(lon);
  free(lat);
  free(area);
  free(press);
  free(np);
  free(ixs);
  free(iys);
  free(izs);
```

```
}
```

Here is the call graph for this function:



5.23.3.76 write_grid_asc()

```
void write_grid_asc (
    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )
```

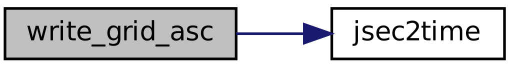
Write gridded data in ASCII format.

Definition at line 5977 of file libtrac.c.

```
               {

  FILE *in, *out;

  char line[LEN];

  /* Check if gnuplot output is requested... */
  if (ctl->grid_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... */
    double r;
    int year, mon, day, hour, min, sec;
    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 {
```

```

    /* 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 depth [km]\n"
          "# $7 = number of particles [1]\n"
          "# $8 = column density (implicit) [kg/m^2]\n"
          "# $9 = volume mixing ratio (implicit) [ppv]\n"
          "# $10 = volume mixing ratio (explicit) [ppv]\n\n");

  /* Write data... */
  for (int ix = 0; ix < ctl->grid_nx; ix++) {
    if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
      fprintf(out, "\n");
    for (int iy = 0; iy < ctl->grid_ny; iy++) {
      if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
        fprintf(out, "\n");
      for (int iz = 0; iz < ctl->grid_nz; iz++) {
        int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
        if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
          fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
                  t, z[iz], lon[ix], lat[iy], area[iy], dz,
                  np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
      }
    }
  }

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

Here is the call graph for this function:



5.23.3.77 write_grid_nc()

```
void write_grid_nc (
    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )
```

Write gridded data in netCDF format.

Definition at line 6063 of file libtrac.c.

```
            {

  double *help;

  int *help2, ncid, dimid[10], varid;

  size_t start[2], count[2];

  /* Allocate... */
  ALLOC(help, double,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
  ALLOC(help2, int,
        ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);

  /* Create file... */
  nc_create(filename, NC_CLOBBER, &ncid);

  /* Define dimensions... */
  NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
  NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[1]));
  NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
  NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[3]));
  NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));

  /* Define variables and their attributes... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
             "seconds since 2000-01-01 00:00:00 UTC");
  NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[1], "altitude", "km");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[3], "longitude", "degrees_east");
  NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[1], "layer depth", "km");
  NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
  NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
  NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
             "volume mixing ratio (implicit)", "ppv");
  NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
             "volume mixing ratio (explicit)", "ppv");
  NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");

  /* End definitions... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC_PUT_DOUBLE("time", &t, 0);
  NC_PUT_DOUBLE("lon", lon, 0);
  NC_PUT_DOUBLE("lat", lat, 0);
  NC_PUT_DOUBLE("z", z, 0);
  NC_PUT_DOUBLE("area", area, 0);
  NC_PUT_DOUBLE("dz", &dz, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          cd[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("cd", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          vmr_impl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("vmr_impl", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          vmr_expl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_DOUBLE("vmr_expl", help, 0);

  for (int ix = 0; ix < ctl->grid_nx; ix++)
    for (int iy = 0; iy < ctl->grid_ny; iy++)
      for (int iz = 0; iz < ctl->grid_nz; iz++)
        help2[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
          np[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
  NC_PUT_INT("np", help2, 0);

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

  /* Free... */
  free(help);
  free(help2);
}
```

5.23.3.78 write_met()

```
int write_met (
    char * filename,
    ctl_t * ctl,
    met_t * met )
```

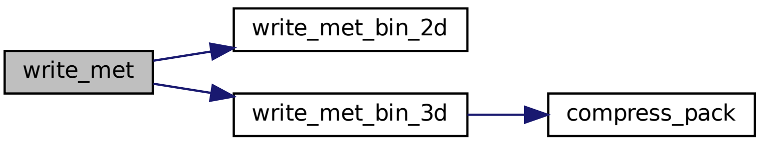
Read meteo data file.

Definition at line 6163 of file libtrac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);

  /* Write info... */
  LOG(1, "Write meteo data: %s", filename);

  /* Check compression flags... */
#ifndef ZFP
  if (ctl->met_type == 3)
    ERRMSG("zfp compression not supported!");
#endif
#ifndef ZSTD
  if (ctl->met_type == 4)
    ERRMSG("zstd compression not supported!");
#endif

  /* Write binary... */
  if (ctl->met_type >= 1 && ctl->met_type <= 4) {

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

    /* Write type of binary data... */
    FWRITE(&ctl->met_type, int,
           1,
           out);

    /* Write version of binary data... */
    int version = 100;
    FWRITE(&version, int,
           1,
           out);

    /* Write grid data... */
    FWRITE(&met->time, double,
           1,
           out);
    FWRITE(&met->nx, int,
           1,
           out);
    FWRITE(&met->ny, int,
           1,
           out);
    FWRITE(&met->np, int,
           1,
           out);
    FWRITE(met->lon, double,
             (size_t) met->nx,
           out);
    FWRITE(met->lat, double,
             (size_t) met->ny,
           out);
    FWRITE(met->p, double,
             (size_t) met->np,
           out);

    /* Write surface data... */
    write_met_bin_2d(out, met, met->ps, "PS");
    write_met_bin_2d(out, met, met->ts, "TS");
    write_met_bin_2d(out, met, met->zs, "ZS");
    write_met_bin_2d(out, met, met->us, "US");
    write_met_bin_2d(out, met, met->vs, "VS");
    write_met_bin_2d(out, met, met->pbl, "PBL");
    write_met_bin_2d(out, met, met->pt, "PT");
    write_met_bin_2d(out, met, met->tt, "TT");
    write_met_bin_2d(out, met, met->zt, "ZT");
    write_met_bin_2d(out, met, met->h2ot, "H2OT");
```

```
    write_met_bin_2d(out, met, met->pct, "PCT");
    write_met_bin_2d(out, met, met->pcb, "PCB");
    write_met_bin_2d(out, met, met->cl, "CL");
    write_met_bin_2d(out, met, met->plcl, "PLCL");
    write_met_bin_2d(out, met, met->plfc, "PLFC");
    write_met_bin_2d(out, met, met->pel, "PEL");
    write_met_bin_2d(out, met, met->cape, "CAPE");
    write_met_bin_2d(out, met, met->cin, "CIN");

    /* Write level data... */
    write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
    write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
    write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
    write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
    write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
    write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
    write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
    write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
    write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
    write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);

    /* Write final flag... */
    int final = 999;
    FWRITE(&final, int,
           1,
           out);

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

  return 0;
}
```

Here is the call graph for this function:



5.23.3.79 write_met_bin_2d()

```
void write_met_bin_2d (
        FILE * out,
        met_t * met,
        float var[EX][EY],
        char * varname )
```

Write 2-D meteo variable.

Definition at line 6273 of file libtrac.c.

```
                        {

  float *help;

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

  /* Copy data... */
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];
```

```

  /* Write uncompressed data... */
  LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
  FWRITE(help, float,
         (size_t) (met->nx * met->ny),
         out);

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

5.23.3.80 write_met_bin_3d()

```
void write_met_bin_3d (
    FILE * out,
    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP],
    char * varname,
    int precision,
    double tolerance )
```

Write 3-D meteo variable.

Definition at line 6302 of file libtrac.c.

```
                         {

  float *help;

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

  /* Copy data... */
#pragma omp parallel for default(shared) collapse(2)
  for (int ix = 0; ix < met->nx; ix++)
    for (int iy = 0; iy < met->ny; iy++)
      for (int ip = 0; ip < met->np; ip++)
        help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];

  /* Write uncompressed data... */
  if (ctl->met_type == 1) {
    LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
    FWRITE(help, float,
           (size_t) (met->nx * met->ny * met->np),
           out);
  }

  /* Write packed data... */
  else if (ctl->met_type == 2)
    compress_pack(varname, help, (size_t) (met->ny * met->nx),
                  (size_t) met->np, 0, out);

  /* Write zfp data... */
#ifdef ZFP
  else if (ctl->met_type == 3)
    compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
                 tolerance, 0, out);
#endif

  /* Write zstd data... */
#ifdef ZSTD
  else if (ctl->met_type == 4)
    compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
                  out);
#endif

  /* Unknown method... */
  else {
    ERRMSG("MET_TYPE not supported!");
    LOG(3, "%d %g", precision, tolerance);
  }

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

Here is the call graph for this function:

5.23.3.81 write_prof()

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

Write profile data.

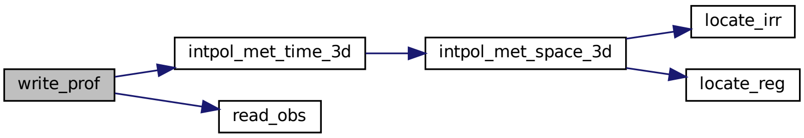
Definition at line 6363 of file libtrac.c.

```
              {

  static FILE *out;

  static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
    dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;

  static int nobs, *obscount, ip, okay;

  /* Set timer... */
  SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);

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

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

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

    /* Allocate... */
    ALLOC(lon, double,
          ctl->prof_nx);
    ALLOC(lat, double,
          ctl->prof_ny);
    ALLOC(area, double,
          ctl->prof_ny);
    ALLOC(z, double,
          ctl->prof_nz);
    ALLOC(press, double,
          ctl->prof_nz);
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
```

```
    read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create new output file... */
    LOG(1, "Write profile data: %s", 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"
            "# $11 = number of observations\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 vertical coordinates... */
    for (int iz = 0; iz < ctl->prof_nz; iz++) {
      z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
      press[iz] = P(z[iz]);
    }

    /* Set horizontal coordinates... */
    for (int ix = 0; ix < ctl->prof_nx; ix++)
      lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
    for (int iy = 0; iy < ctl->prof_ny; iy++) {
      lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
      area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
        * cos(lat[iy] * M_PI / 180.);
    }
  }

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

  /* Allocate... */
  ALLOC(mass, double,
        ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
  ALLOC(obsmean, double,
        ctl->prof_nx * ctl->prof_ny);
  ALLOC(obscount, int,
        ctl->prof_nx * ctl->prof_ny);

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Check observation data... */
    if (!isfinite(robs[i]))
      continue;

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

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

    /* Get mean observation index... */
    int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
    obsmean[idx] += robs[i];
    obscount[idx]++;
  }

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

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

```

    /* Get indices... */
    int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
    int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
    int 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... */
    int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
    mass[idx] += atm->q[ctl->qnt_m][ip];
  }

  /* Extract profiles... */
  for (int ix = 0; ix < ctl->prof_nx; ix++)
    for (int iy = 0; iy < ctl->prof_ny; iy++) {
      int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
      if (obscount[idx2] > 0) {

        /* Check profile... */
        okay = 0;
        for (int iz = 0; iz < ctl->prof_nz; iz++) {
          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
          if (mass[idx3] > 0) {
            okay = 1;
            break;
          }
        }
        if (!okay)
          continue;

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

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

          /* Get temperature, water vapor, and ozone... */
          INTPOL_INIT;
          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
                             lon[ix], lat[iy], &temp, ci, cw, 1);
          intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
                             lon[ix], lat[iy], &h2o, ci, cw, 0);
          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
                             lon[ix], lat[iy], &o3, ci, cw, 0);

          /* Calculate volume mixing ratio... */
          int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
          vmr = MA / ctl->molmass * mass[idx3]
            / (RHO(press[iz], temp) * area[iy] * dz * 1e9);

          /* Write output... */
          fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %d\n",
                  t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
                  obsmean[idx2] / obscount[idx2], obscount[idx2]);
        }
      }
    }

  /* Free... */
  free(mass);
  free(obsmean);
  free(obscount);

  /* Finalize... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(lon);
    free(lat);
    free(area);
    free(z);
    free(press);
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}
```

Here is the call graph for this function:



5.23.3.82 write_sample()

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

Write sample data.

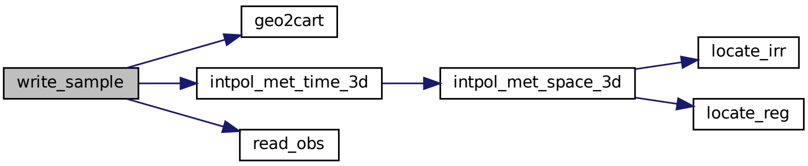
Definition at line 6591 of file libtrac.c.

```
            {

  static FILE *out;

  static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;

  static int nobs;

  /* Set timer... */
  SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);

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

    /* Allocate... */
    ALLOC(rt, double,
          NOBS);
    ALLOC(rz, double,
          NOBS);
    ALLOC(rlon, double,
          NOBS);
    ALLOC(rlat, double,
          NOBS);
    ALLOC(robs, double,
          NOBS);

    /* Read observation data... */
    read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);

    /* Create output file... */
    LOG(1, "Write sample data: %s", 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 = surface area [km^2]\n"
            "# $6 = layer depth [km]\n"
            "# $7 = number of particles [1]\n"
            "# $8 = column density [kg/m^2]\n"
            "# $9 = volume mixing ratio [ppv]\n"
            "# $10 = observed BT index [K]\n\n");

```

```
    /* Set latitude range, squared radius, and area... */
    dlat = DY2DEG(ctl->sample_dx);
    rmax2 = SQR(ctl->sample_dx);
    area = M_PI * rmax2;
  }

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

  /* Loop over observations... */
  for (int i = 0; i < nobs; i++) {

    /* Check time... */
    if (rt[i] < t0)
      continue;
    else if (rt[i] >= t1)
      break;

    /* Calculate Cartesian coordinates... */
    double x0[3];
    geo2cart(0, rlon[i], rlat[i], x0);

    /* Set pressure range... */
    double rp = P(rz[i]);
    double ptop = P(rz[i] + ctl->sample_dz);
    double pbot = P(rz[i] - ctl->sample_dz);

    /* Init... */
    double mass = 0;
    int np = 0;

    /* Loop over air parcels... */
#pragma omp parallel for default(shared) reduction(+:mass,np)
    for (int ip = 0; ip < atm->np; ip++) {

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

      /* Check latitude... */
      if (fabs(rlat[i] - atm->lat[ip]) > dlat)
        continue;

      /* Check horizontal distance... */
      double x1[3];
      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
      if (DIST2(x0, x1) > rmax2)
        continue;

      /* Check pressure... */
      if (ctl->sample_dz > 0)
        if (atm->p[ip] > pbot || atm->p[ip] < ptop)
          continue;

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

    /* Calculate column density... */
    double cd = mass / (1e6 * area);

    /* Calculate volume mixing ratio... */
    double vmr = 0;
    if (ctl->molmass > 0 && ctl->sample_dz > 0) {
      if (mass > 0) {

        /* Get temperature... */
        double temp;
        INTPOL_INIT;
        intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
                           rlon[i], rlat[i], &temp, ci, cw, 1);

        /* Calculate volume mixing ratio... */
        vmr = MA / ctl->molmass * mass
          / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
      }
    } else
      vmr = GSL_NAN;

    /* Write output... */
    fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
            rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
  }

```

```
  /* Finalize...... */
  if (t == ctl->t_stop) {

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

    /* Free... */
    free(rt);
    free(rz);
    free(rlon);
    free(rlat);
    free(robs);
  }
}
```

Here is the call graph for this function:



5.23.3.83 write_station()

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

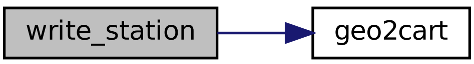
Write station data.

Definition at line 6748 of file libtrac.c.

```
            {

  static FILE *out;

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

  /* Set timer... */
  SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);

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

    /* Write info... */
    LOG(1, "Write station data: %s", 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... */
  double t0 = t - 0.5 * ctl->dt_mod;
  double 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 time range for station output... */
    if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_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:



5.24 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) 2013-2023 Forschungszentrum Juelich GmbH
```

```
*/

#ifndef LIBTRAC_H
#define LIBTRAC_H

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

#include <ctype.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.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

#ifdef ZFP
#include "zfp.h"
#endif

#ifdef ZSTD
#include "zstd.h"
#endif

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

#ifndef CPD
#define CPD 1003.5
#endif

#ifndef EPS
#define EPS (MH2O / MA)
#endif

#ifndef G0
#define G0 9.80665
#endif

#ifndef H0
#define H0 7.0
#endif

#ifndef LV
#define LV 2501000.
#endif

#ifndef KB
#define KB 1.3806504e-23
#endif

#ifndef MA
#define MA 28.9644
#endif

#ifndef MH2O
#define MH2O 18.01528
#endif

#ifndef MO3
#define MO3 48.00
#endif

#ifndef P0
#define P0 1013.25
#endif

#ifndef RA
```

```
#define RA (1e3 * RI / MA)
#endif

#ifndef RE
#define RE 6367.421
#endif

#ifndef RI
#define RI 8.3144598
#endif

#ifndef T0
#define T0 273.15
#endif

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

#ifndef LEN
#define LEN 5000
#endif

#ifndef NP
#define NP 10000000
#endif

#ifndef NQ
#define NQ 15
#endif

#ifndef NCSI
#define NCSI 1000000
#endif

#ifndef EP
#define EP 140
#endif

#ifndef EX
#define EX 1201
#endif

#ifndef EY
#define EY 601
#endif

#ifndef NENS
#define NENS 2000
#endif

#ifndef NOBS
#define NOBS 10000000
#endif

#ifndef NTHREADS
#define NTHREADS 512
#endif

#ifndef CY
#define CY 250
#endif

#ifndef CP
#define CP 60
#endif

#ifndef CT
#define CT 12
#endif

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

#ifdef _OPENACC
#define ALLOC(ptr, type, n)                                  \
  if(acc_get_num_devices(acc_device_nvidia) <= 0)            \
    ERRMSG("Not running on a GPU device!");                  \
  if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)          \
    ERRMSG("Out of memory!");
#else
#define ALLOC(ptr, type, n)                                   \
  if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)           \
    ERRMSG("Out of memory!");
#endif

```

```
#define ARRAY_2D(ix, iy, ny)                          \
  ((ix) * (ny) + (iy))

#define ARRAY_3D(ix, iy, ny, iz, nz)                  \
  (((ix)*(ny) + (iy)) * (nz) + (iz))

#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 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 INTPOL_INIT                                           \
  double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};

#define INTPOL_2D(var, init)                                          \
  intpol_met_time_2d(met0, met0->var, met1, met1->var,                \
                     atm->time[ip], atm->lon[ip], atm->lat[ip],       \
                     &var, ci, cw, init);

#define INTPOL_3D(var, init)                                          \
  intpol_met_time_3d(met0, met0->var, met1, met1->var,                \
                     atm->time[ip], atm->p[ip],                       \
                     atm->lon[ip], atm->lat[ip],                      \
                     &var, ci, cw, init);

#define INTPOL_SPACE_ALL(p, lon, lat) {                               \
  intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1);       \
  intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0);       \
  intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0);       \
  intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0);       \
  intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0);       \
  intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0);     \
  intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0);   \
  intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0);     \
  intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0);   \
  intpol_met_space_3d(met, met->iwc, p, 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->ts, lon, lat, &ts, ci, cw, 0);        \
  intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0);        \
  intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0);        \
  intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0);        \
  intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0);      \
  intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);        \
  intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0);        \
  intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0);        \
  intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0);    \
  intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0);      \
  intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0);      \
  intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0);        \
  intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0);    \
  intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0);    \
```

```
  intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0);        \
  intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0);      \
  intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0);        \
  }

#define INTPOL_TIME_ALL(time, p, lon, lat) {                             \
  intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
  intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
  intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->cl, met1, met1->cl, time, lon, lat, &cl, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
  intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0); \
  }

#define LAPSE(p1, t1, p2, t2)                                            \
  (1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1))                         \
   * ((p2) + (p1)) / ((p2) - (p1)))

#define LIN(x0, y0, x1, y1, x)                      \
  ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))

#define NC(cmd) {                                  \
  int nc_result=(cmd);                             \
  if(nc_result!=NC_NOERR)                          \
    ERRMSG("%s", nc_strerror(nc_result));          \
}

#define NC_DEF_VAR(varname, type, ndims, dims, long_name, units) {       \
    NC(nc_def_var(ncid, varname, type, ndims, dims, &varid));            \
    NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name)); \
    NC(nc_put_att_text(ncid, varid, "units", strlen(units), units));     \
  }

#define NC_GET_DOUBLE(varname, ptr, force) {                     \
    if(force) {                                                  \
      NC(nc_inq_varid(ncid, varname, &varid));                   \
      NC(nc_get_var_double(ncid, varid, ptr));                   \
    } else {                                                     \
      if(nc_inq_varid(ncid, varname, &varid) == NC_NOERR) {      \
        NC(nc_get_var_double(ncid, varid, ptr));                 \
      } else                                                     \
        WARN("netCDF variable %s is missing!", varname);         \
    }                                                            \
  }

#define NC_INQ_DIM(dimname, ptr, min, max) {              \
    int dimid; size_t naux;                               \
    NC(nc_inq_dimid(ncid, dimname, &dimid));              \
    NC(nc_inq_dimlen(ncid, dimid, &naux));                \
    *ptr = (int)naux;                                     \
    if ((*ptr) < (min) || (*ptr) > (max))                 \
      ERRMSG("Dimension %s is out of range!", dimname); \
  }

#define NC_PUT_DOUBLE(varname, ptr, hyperslab) {                 \
    NC(nc_inq_varid(ncid, varname, &varid));                     \
    if(hyperslab) {                                              \
      NC(nc_put_vara_double(ncid, varid, start, count, ptr));    \
    } else {                                                     \
      NC(nc_put_var_double(ncid, varid, ptr));                   \
    }                                                            \
  }

#define NC_PUT_INT(varname, ptr, hyperslab) {                    \
    NC(nc_inq_varid(ncid, varname, &varid));                     \
```

```
    if(hyperslab) {                                          \
      NC(nc_put_vara_int(ncid, varid, start, count, ptr));   \
    } else {                                                 \
      NC(nc_put_var_int(ncid, varid, ptr));                  \
    }                                                        \
  }

#define NC_PUT_ATT(varname, attname, text) {                         \
    NC(nc_inq_varid(ncid, varname, &varid));                         \
    NC(nc_put_att_text(ncid, varid, attname, strlen(text), text));   \
  }

#define NC_PUT_ATT_GLOBAL(attname, text)                             \
  NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));

#define NC_PUT_FLOAT(varname, ptr, hyperslab) {                      \
    NC(nc_inq_varid(ncid, varname, &varid));                         \
    if(hyperslab) {                                                  \
      NC(nc_put_vara_float(ncid, varid, start, count, ptr));         \
    } else {                                                         \
      NC(nc_put_var_float(ncid, varid, ptr));                        \
    }                                                                \
  }

#define NN(x0, y0, x1, y1, x)                                        \
  (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))

#define NORM(a) \
  sqrt(DOTP(a, a))

#define P(z)                                                         \
  (P0 * exp(-(z) / H0))

#define PSAT(t)                                                      \
  (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))

#define PSICE(t)                                                     \
  (6.112 * exp(22.46 * ((t) - T0) / (272.62 + (t) - T0)))

#define PW(p, h2o)                                                   \
  ((p) * GSL_MAX((h2o), 0.1e-6)                                      \
   / (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))

#define RH(p, t, h2o)                                                \
  (PW(p, h2o) / PSAT(t) * 100.)

#define RHICE(p, t, h2o)                                             \
  (PW(p, h2o) / PSICE(t) * 100.)

#define RHO(p, t)                                                    \
  (100. * (p) / (RA * (t)))

#define SET_ATM(qnt, val)                                            \
  if (ctl->qnt >= 0)                                                 \
    atm->q[ctl->qnt][ip] = val;

#define SET_QNT(qnt, name, longname, unit)                           \
  if (strcasecmp(ctl->qnt_name[iq], name) == 0) {                    \
    ctl->qnt = iq;                                                   \
    sprintf(ctl->qnt_longname[iq], longname);                        \
    sprintf(ctl->qnt_unit[iq], unit);                                \
  } else

#define SH(h2o)                                                      \
  (EPS * GSL_MAX((h2o), 0.1e-6))

#define SQR(x)                                                       \
  ((x)*(x))

#define SWAP(x, y, type)                                             \
  do {type tmp = x; x = y; y = tmp;} while(0);

#define TDEW(p, h2o)                                                 \
  (T0 + 243.12 * log(PW((p), (h2o)) / 6.112)                         \
   / (17.62 - log(PW((p), (h2o)) / 6.112)))

#define TICE(p, h2o)                                                 \
  (T0 + 272.62 * log(PW((p), (h2o)) / 6.112)                         \
   / (22.46 - log(PW((p), (h2o)) / 6.112)))

#define THETA(p, t)                                                  \
  ((t) * pow(1000. / (p), 0.286))

#define THETAVIRT(p, t, h2o)                                         \
  (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))

#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. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))

#define Z(p)                                    \
  (H0 * log(P0 / (p)))

#define ZDIFF(lnp0, t0, h2o0, lnp1, t1, h2o1)                          \
  (RI / MA / G0 * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1)))     \
   * ((lnp0) - (lnp1)))

#define ZETA(ps, p, t)                                                 \
  (((p) / (ps) <= 0.3 ? 1. :                                           \
    sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3)))                   \
   * THETA((p), (t)))

/* ------------------------------------------------------------
   Log messages...
   ------------------------------------------------------------ */

#ifndef LOGLEV
#define LOGLEV 2
#endif

#define LOG(level, ...) {                                              \
    if(level >= 2)                                                     \
      printf("  ");                                                    \
    if(level <= LOGLEV) {                                              \
      printf(__VA_ARGS__);                                             \
      printf("\n");                                                    \
    }                                                                  \
  }

#define WARN(...) {                                                    \
    printf("\nWarning (%s, %s, l%d): ", __FILE__, __func__, __LINE__); \
    LOG(0, __VA_ARGS__);                                               \
  }

#define ERRMSG(...) {                                                  \
    printf("\nError (%s, %s, l%d): ", __FILE__, __func__, __LINE__);   \
    LOG(0, __VA_ARGS__);                                               \
    exit(EXIT_FAILURE);                                                \
  }

#define PRINT(format, var)                                             \
  printf("Print (%s, %s, l%d): %s= "format"\n",                        \
         __FILE__, __func__, __LINE__, #var, var);

/* ------------------------------------------------------------
   Timers...
   ------------------------------------------------------------ */

#define NTIMER 100

#define PRINT_TIMERS                            \
  timer("END", "END", 1);

#define SELECT_TIMER(id, group, color) {                               \
    NVTX_POP;                                                          \
    NVTX_PUSH(id, color);                                              \
    timer(id, group, 0);                                               \
  }

#define START_TIMERS                            \
  NVTX_PUSH("START", NVTX_CPU);

#define STOP_TIMERS                             \
  NVTX_POP;

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

#ifdef NVTX
#include "nvToolsExt.h"

#define NVTX_CPU 0xFFADD8E6

#define NVTX_GPU 0xFF00008B

#define NVTX_H2D 0xFFFFFF00

#define NVTX_D2H 0xFFFF8800
```

```c
#define NVTX_READ 0xFFFFCCCB
#define NVTX_WRITE 0xFF8B0000
#define NVTX_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 NVTX_POP {                               \
    nvtxRangePop();                              \
  }
#else
/* Empty definitions of NVTX_PUSH and NVTX_POP... */
#define NVTX_PUSH(range_title, range_color) {}
#define NVTX_POP {}
#endif
/* ------------------------------------------------------------
   Thrust...
   ------------------------------------------------------------ */
void thrustSortWrapper(
  double *__restrict__ c,
  int n,
  int *__restrict__ index);
/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */
typedef struct {
  int vert_coord_ap;
  int vert_coord_met;
  int vert_vel;
  int clams_met_data;
  size_t chunkszhint;
  int read_mode;
  int nq;
  char qnt_name[NQ][LEN];
  char qnt_longname[NQ][LEN];
  char qnt_unit[NQ][LEN];
  char qnt_format[NQ][LEN];
  int qnt_idx;
  int qnt_ens;
  int qnt_stat;
  int qnt_m;
  int qnt_vmr;
  int qnt_rp;
  int qnt_rhop;
  int qnt_ps;
  int qnt_ts;
  int qnt_zs;
  int qnt_us;
  int qnt_vs;
```

```
int qnt_pbl;
int qnt_pt;
int qnt_tt;
int qnt_zt;
int qnt_h2ot;
int qnt_z;
int qnt_p;
int qnt_t;
int qnt_rho;
int qnt_u;
int qnt_v;
int qnt_w;
int qnt_h2o;
int qnt_o3;
int qnt_lwc;
int qnt_iwc;
int qnt_pct;
int qnt_pcb;
int qnt_cl;
int qnt_plcl;
int qnt_plfc;
int qnt_pel;
int qnt_cape;
int qnt_cin;
int qnt_hno3;
int qnt_oh;
int qnt_vmrimpl;
int qnt_mloss_oh;
int qnt_mloss_h2o2;
int qnt_mloss_wet;
int qnt_mloss_dry;
int qnt_mloss_decay;
int qnt_psat;
int qnt_psice;
int qnt_pw;
int qnt_sh;
int qnt_rh;
int qnt_rhice;
int qnt_theta;
int qnt_zeta;
int qnt_tvirt;
int qnt_lapse;
int qnt_vh;
int qnt_vz;
```

```
int qnt_pv;
int qnt_tdew;
int qnt_tice;
int qnt_tsts;
int qnt_tnat;
int direction;
double t_start;
double t_stop;
double dt_mod;
char metbase[LEN];
double dt_met;
int met_type;
int met_nc_scale;
int met_dx;
int met_dy;
int met_dp;
int met_sx;
int met_sy;
int met_sp;
double met_detrend;
int met_np;
double met_p[EP];
int met_geopot_sx;
int met_geopot_sy;
int met_relhum;
int met_tropo;
double met_tropo_lapse;
int met_tropo_nlev;
double met_tropo_lapse_sep;
int met_tropo_nlev_sep;
double met_tropo_pv;
double met_tropo_theta;
int met_tropo_spline;
int met_cloud;
double met_cloud_min;
double met_dt_out;
int met_cache;
double sort_dt;
int isosurf;
char balloon[LEN];
int advect;
int reflect;
double turb_dx_trop;
```

```
  double turb_dx_strat;

  double turb_dz_trop;

  double turb_dz_strat;

  double turb_mesox;

  double turb_mesoz;

  double conv_cape;

  double conv_cin;

  double conv_dt;

  int conv_mix;

  int conv_mix_bot;

  int conv_mix_top;

  double bound_mass;

  double bound_mass_trend;

  double bound_vmr;

  double bound_vmr_trend;

  double bound_lat0;

  double bound_lat1;

  double bound_p0;

  double bound_p1;

  double bound_dps;

  double bound_dzs;

  double bound_zetas;

  int bound_pbl;

  char species[LEN];

  double molmass;

  double tdec_trop;

  double tdec_strat;

  char clim_oh_filename[LEN];

  char clim_h2o2_filename[LEN];

  int oh_chem_reaction;

  double oh_chem[4];

  double oh_chem_beta;

  double h2o2_chem_cc;

  int h2o2_chem_reaction;

  int chemgrid_nz;

  double chemgrid_z0;

  double chemgrid_z1;

  int chemgrid_nx;

  double chemgrid_lon0;

  double chemgrid_lon1;

  int chemgrid_ny;

  double chemgrid_lat0;

  double chemgrid_lat1;

  double dry_depo_dp;
```

```
double dry_depo_vdep;
double wet_depo_pre[2];
double wet_depo_bc_a;
double wet_depo_bc_b;
double wet_depo_ic_a;
double wet_depo_ic_b;
double wet_depo_ic_h[3];
double wet_depo_bc_h[2];
double wet_depo_ic_ret_ratio;
double wet_depo_bc_ret_ratio;
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 ens_basename[LEN];
double ens_dt_out;
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;
  int grid_type;
  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 sample_basename[LEN];
  char sample_obsfile[LEN];
  double sample_dx;
  double sample_dz;
  char stat_basename[LEN];
  double stat_lon;
  double stat_lat;
  double stat_r;
  double stat_t0;
  double stat_t1;
} ctl_t;
typedef struct {
  int np;
  double time[NP];
  double p[NP];
  double zeta[NP];
  double lon[NP];
  double lat[NP];
  double q[NQ][NP];
} atm_t;
typedef struct {
  double iso_var[NP];
  double iso_ps[NP];
  double iso_ts[NP];
  int iso_n;
  float uvwp[NP][3];
} cache_t;
typedef struct {
```

```
  int tropo_ntime;
  int tropo_nlat;
  double tropo_time[12];
  double tropo_lat[73];
  double tropo[12][73];
  int hno3_ntime;
  int hno3_nlat;
  int hno3_np;
  double hno3_time[12];
  double hno3_lat[18];
  double hno3_p[10];
  double hno3[12][18][10];
  int oh_ntime;
  int oh_nlat;
  int oh_np;
  double oh_time[CT];
  double oh_lat[CY];
  double oh_p[CP];
  double oh[CT][CP][CY];
  int h2o2_ntime;
  int h2o2_nlat;
  int h2o2_np;
  double h2o2_time[CT];
  double h2o2_lat[CY];
  double h2o2_p[CP];
  double h2o2[CT][CP][CY];
} clim_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 ts[EX][EY];
  float zs[EX][EY];
  float us[EX][EY];
  float vs[EX][EY];
  float pbl[EX][EY];
  float pt[EX][EY];
  float tt[EX][EY];
```

```
  float zt[EX][EY];

  float h2ot[EX][EY];

  float pct[EX][EY];

  float pcb[EX][EY];

  float cl[EX][EY];

  float plcl[EX][EY];

  float plfc[EX][EY];

  float pel[EX][EY];

  float cape[EX][EY];

  float cin[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];

  float patp[EX][EY][EP];

  float zeta[EX][EY][EP];

  float zeta_dot[EX][EY][EP];

#ifdef UVW
  float uvw[EX][EY][EP][3];
#endif

} met_t;

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

double buoyancy_frequency(
  double p0,
  double t0,
  double p1,
  double t1);

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(
  clim_t * clim,
  double t,
  double lat,
  double p);

void clim_hno3_init(
  clim_t * clim);
```

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

#ifdef _OPENACC
#pragma acc routine (clim_oh_diurnal)
#endif
double clim_oh_diurnal(
  ctl_t * ctl,
  clim_t * clim,
  double t,
  double p,
  double lon,
  double lat);

void clim_oh_init(
  ctl_t * ctl,
  clim_t * clim);

double clim_oh_init_help(
  double beta,
  double time,
  double lat);

#ifdef _OPENACC
#pragma acc routine (clim_h2o2)
#endif
double clim_h2o2(
  clim_t * clim,
  double t,
  double lat,
  double p);

void clim_h2o2_init(
  ctl_t * ctl,
  clim_t * clim);

#ifdef _OPENACC
#pragma acc routine (clim_tropo)
#endif
double clim_tropo(
  clim_t * clim,
  double t,
  double lat);

void clim_tropo_init(
  clim_t * clim);

void compress_pack(
  char *varname,
  float *array,
  size_t nxy,
  size_t nz,
  int decompress,
  FILE * inout);

#ifdef ZFP
void compress_zfp(
  char *varname,
  float *array,
  int nx,
  int ny,
  int nz,
  int precision,
  double tolerance,
  int decompress,
  FILE * inout);
#endif

#ifdef ZSTD
void compress_zstd(
  char *varname,
  float *array,
  size_t n,
  int decompress,
  FILE * inout);
#endif

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,
  clim_t * clim,
  double t,
  met_t ** met0,
  met_t ** met1);

void get_met_help(
  ctl_t * ctl,
  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 UVW
#ifdef _OPENACC
#pragma acc routine (intpol_met_space_uvw)
#endif
void intpol_met_space_uvw(
  met_t * met,
  double p,
  double lon,
  double lat,
  double *u,
  double *v,
  double *w,
  int *ci,
  double *cw,
  int init);
#endif

#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);

#ifdef UVW
#ifdef _OPENACC
#pragma acc routine (intpol_met_time_uvw)
#endif
void intpol_met_time_uvw(
  met_t * met0,
  met_t * met1,
  double ts,
  double p,
  double lon,
  double lat,
  double *u,
  double *v,
  double *w);
#endif

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

#ifdef _OPENACC
#pragma acc routine (lapse_rate)
#endif
double lapse_rate(
  double t,
  double h2o);

#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);

#ifdef _OPENACC
#pragma acc routine (nat_temperature)
#endif
double nat_temperature(
  double p,
  double h2o,
  double hno3);

void quicksort(
  double arr[],
  int brr[],
  int low,
  int high);
```

```
int quicksort_partition(
  double arr[],
  int brr[],
  int low,
  int high);

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

int read_atm_asc(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

int read_atm_bin(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

int read_atm_clams(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

int read_atm_nc(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

void read_clim(
  ctl_t * ctl,
  clim_t * clim);

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

int read_met(
  char *filename,
  ctl_t * ctl,
  clim_t * clim,
  met_t * met);

void read_met_bin_2d(
  FILE * out,
  met_t * met,
  float var[EX][EY],
  char *varname);

void read_met_bin_3d(
  FILE * in,
  ctl_t * ctl,
  met_t * met,
  float var[EX][EY][EP],
  char *varname,
  int precision,
  double tolerance);

void read_met_cape(
  clim_t * clim,
  met_t * met);

void read_met_cloud(
  ctl_t * ctl,
  met_t * met);

void read_met_detrend(
  ctl_t * ctl,
  met_t * met);

void read_met_extrapolate(
  met_t * met);

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

void read_met_grid(
  char *filename,
  int ncid,
  ctl_t * ctl,
  met_t * met);
```

```
void read_met_levels(
  int ncid,
  ctl_t * ctl,
  met_t * met);

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

int read_met_nc_2d(
  int ncid,
  char *varname,
  char *varname2,
  ctl_t * ctl,
  met_t * met,
  float dest[EX][EY],
  float scl,
  int init);

int read_met_nc_3d(
  int ncid,
  char *varname,
  char *varname2,
  ctl_t * ctl,
  met_t * met,
  float dest[EX][EY][EP],
  float scl,
  int init);

void read_met_pbl(
  met_t * met);

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,
  ctl_t * ctl);

void read_met_tropo(
  ctl_t * ctl,
  clim_t * clim,
  met_t * met);

void read_obs(
  char *filename,
  double *rt,
  double *rz,
  double *rlon,
  double *rlat,
  double *robs,
  int *nobs);

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

#ifdef _OPENACC
#pragma acc routine (sedi)
#endif
double sedi(
  double p,
  double T,
  double rp,
  double rhop);

void spline(
  double *x,
  double *y,
  int n,
  double *x2,
  double *y2,
```

```
  int n2,
  int method);

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

#ifdef _OPENACC
#pragma acc routine (sza)
#endif
double sza(
  double sec,
  double lon,
  double lat);

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

void timer(
  const char *name,
  const char *group,
  int output);

#ifdef _OPENACC
#pragma acc routine (tropo_weight)
#endif
double tropo_weight(
  clim_t * clim,
  double t,
  double lat,
  double p);

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

void write_atm_asc(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm,
  double t);

void write_atm_bin(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

void write_atm_clams(
  ctl_t * ctl,
  atm_t * atm,
  double t);

void write_atm_nc(
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

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_grid_asc(
  const char *filename,
  ctl_t * ctl,
  double *cd,
  double *vmr_expl,
  double *vmr_impl,
  double t,
  double *z,
  double *lon,
  double *lat,
  double *area,
  double dz,
  int *np);

void write_grid_nc(
  const char *filename,
  ctl_t * ctl,
  double *cd,
  double *vmr_expl,
  double *vmr_impl,
  double t,
  double *z,
  double *lon,
  double *lat,
  double *area,
  double dz,
  int *np);

int write_met(
  char *filename,
  ctl_t * ctl,
  met_t * met);

void write_met_bin_2d(
  FILE * out,
  met_t * met,
  float var[EX][EY],
  char *varname);

void write_met_bin_3d(
  FILE * out,
  ctl_t * ctl,
  met_t * met,
  float var[EX][EY][EP],
  char *varname,
  int precision,
  double tolerance);

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

void write_sample(
  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);

#endif /* LIBTRAC_H */
```

5.25 met_conv.c File Reference

Convert file format of meteo data files.

```
#include "libtrac.h"
```

Functions

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

5.25.1 Detailed Description

Convert file format of meteo data files.

Definition in file met_conv.c.

5.25.2 Function Documentation

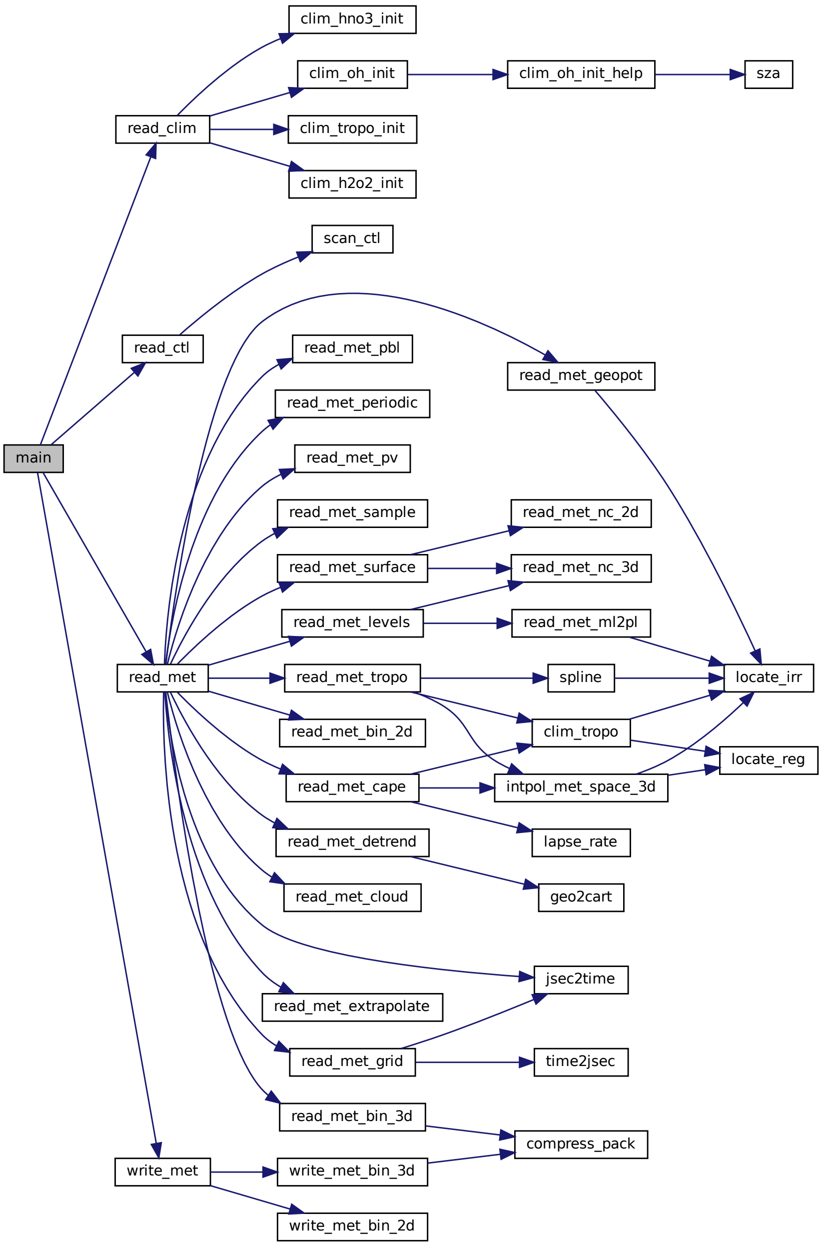
5.25.2.1 main()

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

Definition at line 27 of file met_conv.c.

```
                  {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <met_in> <met_in_type>"
           " <met_out> <met_out_type>");

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

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

  /* Read climatological data... */
  read_clim(&ctl, clim);

  /* Read meteo data... */
  ctl.met_type = atoi(argv[3]);
  if (!read_met(argv[2], &ctl, clim, met))
    ERRMSG("Cannot open file!");

  /* Write meteo data... */
  ctl.met_type = atoi(argv[5]);
  write_met(argv[4], &ctl, met);

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.26 met_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) 2013-2022 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <met_in> <met_in_type>"
           " <met_out> <met_out_type>");

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

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

  /* Read climatological data... */
  read_clim(&ctl, clim);

  /* Read meteo data... */
  ctl.met_type = atoi(argv[3]);
  if (!read_met(argv[2], &ctl, clim, met))
    ERRMSG("Cannot open file!");

  /* Write meteo data... */
  ctl.met_type = atoi(argv[5]);
  write_met(argv[4], &ctl, met);

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

5.27 met_lapse.c File Reference

Calculate lapse rate statistics.

`#include "libtrac.h"`

Macros

- #define LAPSEMIN -20.0

    Lapse rate minimum [K/km.
- #define DLAPSE 0.1

    Lapse rate bin size [K/km].
- #define IDXMAX 400

    Maximum number of histogram bins.

Functions

- int main (int argc, char ∗argv[ ])

5.27.1 Detailed Description

Calculate lapse rate statistics.

Definition in file met_lapse.c.

5.27.2 Macro Definition Documentation

5.27.2.1 LAPSEMIN `#define LAPSEMIN -20.0`

Lapse rate minimum [K/km.

Definition at line 32 of file met_lapse.c.

5.27.2.2 DLAPSE `#define DLAPSE 0.1`

Lapse rate bin size [K/km].

Definition at line 35 of file met_lapse.c.

5.27.2.3 IDXMAX `#define IDXMAX 400`

Maximum number of histogram bins.

Definition at line 38 of file met_lapse.c.

5.27.3 Function Documentation

5.27.3.1 main()

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

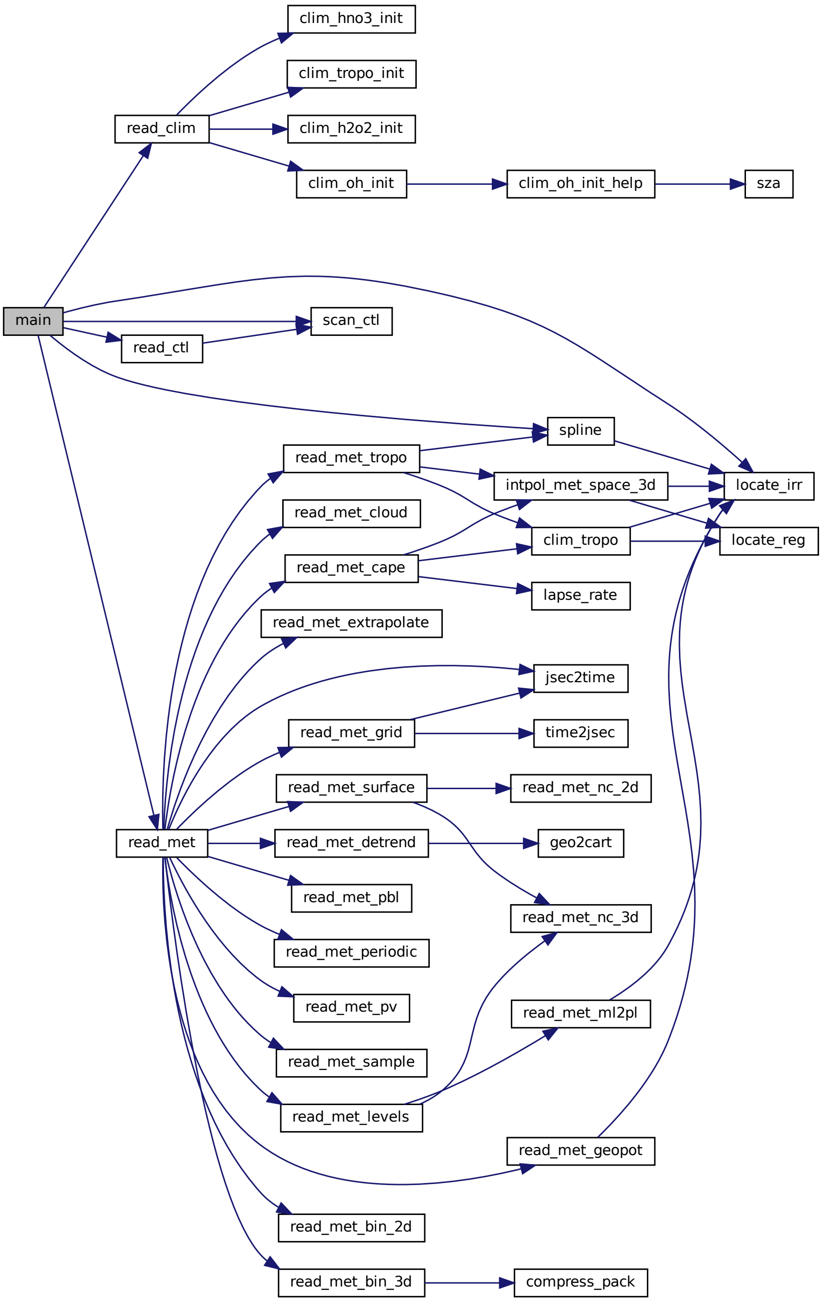
Definition at line 44 of file met_lapse.c.

```
                  {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
    z_mean;

  static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
    nhist_max, nhist_min, nhist_mean, nhist_sig, np;

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <lapse.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
  double lat0 =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
  double lat1 =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
  double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
  double z1 =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
  int intpol =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, met))
      continue;

    /* Get altitude and pressure profiles... */
    for (int iz = 0; iz < met->np; iz++)
      z[iz] = Z(met->p[iz]);
    for (int iz = 0; iz <= 250; iz++) {
      z2[iz] = 0.0 + 0.1 * iz;
      p2[iz] = P(z2[iz]);
    }

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

        /* Check latitude range... */
        if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
          continue;

        /* Interpolate temperature profile... */
        for (int iz = 0; iz < met->np; iz++)
          t[iz] = met->t[ix][iy][iz];
        if (intpol == 1)
          spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
        else
          for (int iz = 0; iz <= 250; iz++) {
            int idx = locate_irr(z, met->np, z2[iz]);
            t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
          }

        /* Loop over vertical levels... */
        for (int iz = 0; iz <= 250; iz++) {

          /* Check height range... */
          if (z2[iz] < z0 || z2[iz] > z1)
            continue;
```

```

        /* Check surface pressure... */
        if (p2[iz] > met->ps[ix][iy])
          continue;

        /* Get mean latitude and height... */
        lat_mean += met->lat[iy];
        z_mean += z2[iz];
        np++;

        /* Get lapse rates within a vertical layer... */
        int nlapse = 0;
        double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
          0, lapse_sig = 0;
        for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
          lapse_max =
            GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
          lapse_min =
            GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
          lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
          lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
          nlapse++;
        }
        lapse_mean /= nlapse;
        lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));

        /* Get histograms... */
        int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_max[idx]++;
          nhist_max++;
        }

        idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_min[idx]++;
          nhist_min++;
        }

        idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_mean[idx]++;
          nhist_mean++;
        }

        idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_sig[idx]++;
          nhist_sig++;
        }
      }
    }
}

/* Create output file... */
LOG(1, "Write lapse rate data: %s", argv[2]);
if (!(out = fopen(argv[2], "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
        "# $1 = mean altitude [km]\n"
        "# $2 = mean latitude [deg]\n"
        "# $3 = lapse rate [K/km]\n"
        "# $4 = counts of maxima per bin\n"
        "# $5 = total number of maxima\n"
        "# $6 = normalized frequency of maxima\n"
        "# $7 = counts of minima per bin\n"
        "# $8 = total number of minima\n"
        "# $9 = normalized frequency of minima\n"
        "# $10 = counts of means per bin\n"
        "# $11 = total number of means\n"
        "# $12 = normalized frequency of means\n"
        "# $13 = counts of sigmas per bin\n"
        "# $14 = total number of sigmas\n"
        "# $15 = normalized frequency of sigmas\n\n");

/* Write data... */
double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
for (int idx = 0; idx < IDXMAX; idx++) {
  nmax_max = GSL_MAX(hist_max[idx], nmax_max);
  nmax_min = GSL_MAX(hist_min[idx], nmax_min);
  nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
  nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
}
for (int idx = 0; idx < IDXMAX; idx++)
  fprintf(out,
```

```
            "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
            z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
            hist_max[idx], nhist_max,
            (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
            nhist_min, (double) hist_min[idx] / (double) nmax_min,
            hist_mean[idx], nhist_mean,
            (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
            nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.28 met_lapse.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) 2013-2021 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

#define LAPSEMIN -20.0

#define DLAPSE 0.1

#define IDXMAX 400

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

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
    z_mean;

  static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
    nhist_max, nhist_min, nhist_mean, nhist_sig, np;

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <lapse.tab> <met0> [ <met1> ... ]");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
  double lat0 =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
  double lat1 =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
  double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
  double z1 =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
  int intpol =
    (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, met))
      continue;

    /* Get altitude and pressure profiles... */
    for (int iz = 0; iz < met->np; iz++)
      z[iz] = Z(met->p[iz]);
    for (int iz = 0; iz <= 250; iz++) {
      z2[iz] = 0.0 + 0.1 * iz;
      p2[iz] = P(z2[iz]);
    }

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

        /* Check latitude range... */
        if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
```

```
        continue;

      /* Interpolate temperature profile... */
      for (int iz = 0; iz < met->np; iz++)
        t[iz] = met->t[ix][iy][iz];
      if (intpol == 1)
        spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
      else
        for (int iz = 0; iz <= 250; iz++) {
          int idx = locate_irr(z, met->np, z2[iz]);
          t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
        }

      /* Loop over vertical levels... */
      for (int iz = 0; iz <= 250; iz++) {

        /* Check height range... */
        if (z2[iz] < z0 || z2[iz] > z1)
          continue;

        /* Check surface pressure... */
        if (p2[iz] > met->ps[ix][iy])
          continue;

        /* Get mean latitude and height... */
        lat_mean += met->lat[iy];
        z_mean += z2[iz];
        np++;

        /* Get lapse rates within a vertical layer... */
        int nlapse = 0;
        double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
          0, lapse_sig = 0;
        for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
          lapse_max =
            GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
          lapse_min =
            GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
          lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
          lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
          nlapse++;
        }
        lapse_mean /= nlapse;
        lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));

        /* Get histograms... */
        int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_max[idx]++;
          nhist_max++;
        }

        idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_min[idx]++;
          nhist_min++;
        }

        idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_mean[idx]++;
          nhist_mean++;
        }

        idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
        if (idx >= 0 && idx < IDXMAX) {
          hist_sig[idx]++;
          nhist_sig++;
        }
      }
    }
}

/* Create output file... */
LOG(1, "Write lapse rate data: %s", argv[2]);
if (!(out = fopen(argv[2], "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
        "# $1 = mean altitude [km]\n"
        "# $2 = mean latitude [deg]\n"
        "# $3 = lapse rate [K/km]\n"
        "# $4 = counts of maxima per bin\n"
        "# $5 = total number of maxima\n"
        "# $6 = normalized frequency of maxima\n"
        "# $7 = counts of minima per bin\n"
```

```
          "# $8 = total number of minima\n"
          "# $9 = normalized frequency of minima\n"
          "# $10 = counts of means per bin\n"
          "# $11 = total number of means\n"
          "# $12 = normalized frequency of means\n"
          "# $13 = counts of sigmas per bin\n"
          "# $14 = total number of sigmas\n"
          "# $15 = normalized frequency of sigmas\n\n");

  /* Write data... */
  double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
  for (int idx = 0; idx < IDXMAX; idx++) {
    nmax_max = GSL_MAX(hist_max[idx], nmax_max);
    nmax_min = GSL_MAX(hist_min[idx], nmax_min);
    nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
    nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
  }
  for (int idx = 0; idx < IDXMAX; idx++)
    fprintf(out,
            "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
            z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
            hist_max[idx], nhist_max,
            (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
            nhist_min, (double) hist_min[idx] / (double) nmax_min,
            hist_mean[idx], nhist_mean,
            (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
            nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

5.29 met_map.c File Reference

Extract map from meteorological data.

```
#include "libtrac.h"
```

Macros

- #define NX 1441

    Maximum number of longitudes.

- #define NY 721

    Maximum number of latitudes.

Functions

- int main (int argc, char ∗argv[ ])

5.29.1 Detailed Description

Extract map from meteorological data.

Definition in file met_map.c.

5.29.2 Macro Definition Documentation

5.29.2.1 NX `#define NX 1441`

Maximum number of longitudes.

Definition at line 32 of file met_map.c.

5.29.2.2 NY `#define NY 721`

Maximum number of latitudes.

Definition at line 35 of file met_map.c.

5.29.3 Function Documentation

5.29.3.1 main()

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

Definition at line 41 of file met_map.c.

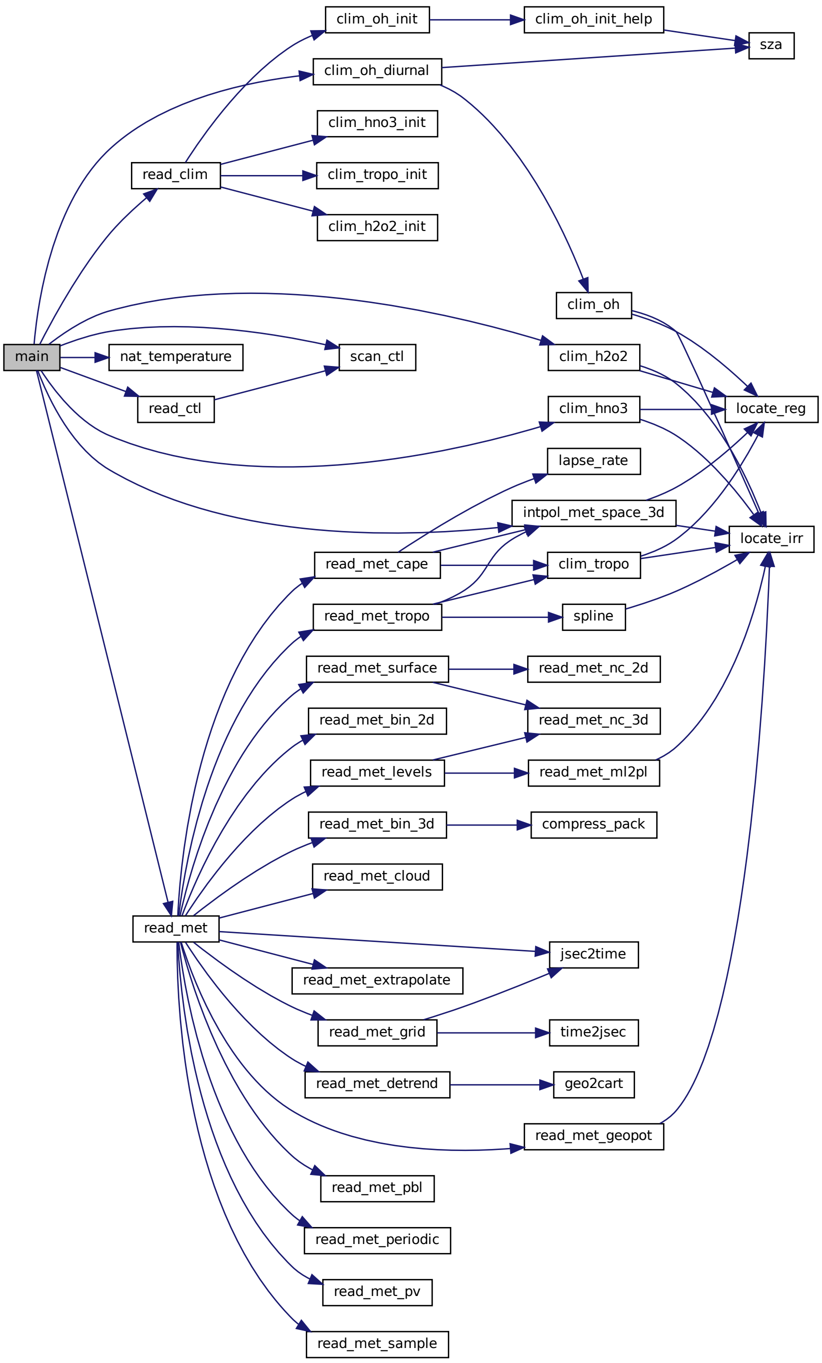
```
                      {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
    zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
    ptm[NX][NY], t, pm[NX][NY], 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],
    hno3m[NX][NY], ohm[NX][NY], h2o2m[NX][NY], tdewm[NX][NY], ticem[NX][NY],
    tnatm[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], pct, pctm[NX][NY], pcb,
    pcbm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
    pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
    rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
    lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, 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++) {

        /* Find pressure level for given theta level... */
        if (theta > 0) {
          ptop = met->p[met->np - 1];
          pbot = met->p[0];
          do {
            p0 = 0.5 * (ptop + pbot);
            intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
                                &t0, ci, cw, 1);
            if (THETA(p0, t0) > theta)
              ptop = p0;
            else
              pbot = p0;
          } while (fabs(ptop - pbot) > 1e-5);
        }

        /* Interpolate meteo data... */
        INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);

        /* Averaging... */
        timem[ix][iy] += met->time;
        zm[ix][iy] += z;
        pm[ix][iy] += p0;
        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;
        tsm[ix][iy] += ts;
        zsm[ix][iy] += zs;
        usm[ix][iy] += us;
        vsm[ix][iy] += vs;
        pblm[ix][iy] += pbl;
        pctm[ix][iy] += pct;
        pcbm[ix][iy] += pcb;
        clm[ix][iy] += cl;
        if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
            && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
          plclm[ix][iy] += plcl;
          plfcm[ix][iy] += plfc;
```

```
            pelm[ix][iy] += pel;
            capem[ix][iy] += cape;
            cinm[ix][iy] += cin;
            npc[ix][iy]++;
          }
          if (gsl_finite(pt)) {
            ptm[ix][iy] += pt;
            ztm[ix][iy] += zt;
            ttm[ix][iy] += tt;
            h2otm[ix][iy] += h2ot;
            npt[ix][iy]++;
          }
          hno3m[ix][iy] += clim_hno3(clim, met->time, lats[iy], p0);
          tnatm[ix][iy] +=
            nat_temperature(p0, h2o, clim_hno3(clim, met->time, lats[iy], p0));
          ohm[ix][iy] +=
            clim_oh_diurnal(&ctl, clim, met->time, p0, lons[ix], lats[iy]);
          h2o2m[ix][iy] += clim_h2o2(clim, met->time, lats[iy], p0);
          rhm[ix][iy] += RH(p0, t, h2o);
          rhicem[ix][iy] += RHICE(p0, t, h2o);
          tdewm[ix][iy] += TDEW(p0, h2o);
          ticem[ix][iy] += TICE(p0, h2o);
          np[ix][iy]++;
        }
    }

    /* Create output file... */
    LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
            "# $16 = surface geopotential height [km]\n"
            "# $17 = surface zonal wind [m/s]\n"
            "# $18 = surface meridional wind [m/s]\n"
            "# $19 = tropopause pressure [hPa]\n"
            "# $20 = tropopause geopotential height [km]\n");
    fprintf(out,
            "# $21 = tropopause temperature [K]\n"
            "# $22 = tropopause water vapor [ppv]\n"
            "# $23 = cloud liquid water content [kg/kg]\n"
            "# $24 = cloud ice water content [kg/kg]\n"
            "# $25 = total column cloud water [kg/m^2]\n"
            "# $26 = cloud top pressure [hPa]\n"
            "# $27 = cloud bottom pressure [hPa]\n"
            "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
            "# $29 = pressure at level of free convection (LFC) [hPa]\n"
            "# $30 = pressure at equilibrium level (EL) [hPa]\n");
    fprintf(out,
            "# $31 = convective available potential energy (CAPE) [J/kg]\n"
            "# $32 = convective inhibition (CIN) [J/kg]\n"
            "# $33 = relative humidity over water [%%]\n"
            "# $34 = relative humidity over ice [%%]\n"
            "# $35 = dew point temperature [K]\n"
            "# $36 = frost point temperature [K]\n"
            "# $37 = NAT temperature [K]\n"
            "# $38 = HNO3 volume mixing ratio [ppv]\n"
            "# $39 = OH concentration [molec/cm^3]\n"
            "# $40 = H2O2 concentration [molec/cm^3]\n");
    fprintf(out,
            "# $41 = boundary layer pressure [hPa]\n"
            "# $42 = number of data points\n"
            "# $43 = number of tropopause data points\n"
            "# $44 = number of CAPE data points\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 %g %g %g %g %g %g %g"
            " %g %g %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
            timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
            lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
            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], tsm[ix][iy] / np[ix][iy],
            zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
            vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
            ztm[ix][iy] / npt[ix][iy], ttm[ix][iy] / npt[ix][iy],
            h2otm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
            iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
            pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
            plclm[ix][iy] / npc[ix][iy], plfcm[ix][iy] / npc[ix][iy],
            pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
            cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
            rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
            ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
            hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
            h2o2m[ix][iy] / np[ix][iy], pblm[ix][iy] / np[ix][iy],
            np[ix][iy], npt[ix][iy], npc[ix][iy]);
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.30 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) 2013-2022 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

#define NX 1441

#define NY 721

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

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
    zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
    ptm[NX][NY], t, pm[NX][NY], 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],
    hno3m[NX][NY], ohm[NX][NY], h2o2m[NX][NY], tdewm[NX][NY], ticem[NX][NY],
    tnatm[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], pct, pctm[NX][NY], pcb,
    pcbm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
    pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
    rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
    lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, 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++) {

      /* Find pressure level for given theta level... */
      if (theta > 0) {
        ptop = met->p[met->np - 1];
        pbot = met->p[0];
        do {
          p0 = 0.5 * (ptop + pbot);
          intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
                              &t0, ci, cw, 1);
          if (THETA(p0, t0) > theta)
            ptop = p0;
          else
            pbot = p0;
        } while (fabs(ptop - pbot) > 1e-5);
      }

      /* Interpolate meteo data... */
      INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);

      /* Averaging... */
      timem[ix][iy] += met->time;
      zm[ix][iy] += z;
      pm[ix][iy] += p0;
      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;
      tsm[ix][iy] += ts;
      zsm[ix][iy] += zs;
      usm[ix][iy] += us;
      vsm[ix][iy] += vs;
      pblm[ix][iy] += pbl;
      pctm[ix][iy] += pct;
      pcbm[ix][iy] += pcb;
      clm[ix][iy] += cl;
      if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
          && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
        plclm[ix][iy] += plcl;
        plfcm[ix][iy] += plfc;
        pelm[ix][iy] += pel;
        capem[ix][iy] += cape;
        cinm[ix][iy] += cin;
        npc[ix][iy]++;
      }
      if (gsl_finite(pt)) {
        ptm[ix][iy] += pt;
        ztm[ix][iy] += zt;
        ttm[ix][iy] += tt;
        h2otm[ix][iy] += h2ot;
        npt[ix][iy]++;
      }
      hno3m[ix][iy] += clim_hno3(clim, met->time, lats[iy], p0);
      tnatm[ix][iy] +=
        nat_temperature(p0, h2o, clim_hno3(clim, met->time, lats[iy], p0));
      ohm[ix][iy] +=
        clim_oh_diurnal(&ctl, clim, met->time, p0, lons[ix], lats[iy]);
      h2o2m[ix][iy] += clim_h2o2(clim, met->time, lats[iy], p0);
      rhm[ix][iy] += RH(p0, t, h2o);
      rhicem[ix][iy] += RHICE(p0, t, h2o);
      tdewm[ix][iy] += TDEW(p0, h2o);
```

```
        ticem[ix][iy] += TICE(p0, h2o);
        np[ix][iy]++;
      }
  }

  /* Create output file... */
  LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
          "# $16 = surface geopotential height [km]\n"
          "# $17 = surface zonal wind [m/s]\n"
          "# $18 = surface meridional wind [m/s]\n"
          "# $19 = tropopause pressure [hPa]\n"
          "# $20 = tropopause geopotential height [km]\n");
  fprintf(out,
          "# $21 = tropopause temperature [K]\n"
          "# $22 = tropopause water vapor [ppv]\n"
          "# $23 = cloud liquid water content [kg/kg]\n"
          "# $24 = cloud ice water content [kg/kg]\n"
          "# $25 = total column cloud water [kg/m^2]\n"
          "# $26 = cloud top pressure [hPa]\n"
          "# $27 = cloud bottom pressure [hPa]\n"
          "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
          "# $29 = pressure at level of free convection (LFC) [hPa]\n"
          "# $30 = pressure at equilibrium level (EL) [hPa]\n");
  fprintf(out,
          "# $31 = convective available potential energy (CAPE) [J/kg]\n"
          "# $32 = convective inhibition (CIN) [J/kg]\n"
          "# $33 = relative humidity over water [%%]\n"
          "# $34 = relative humidity over ice [%%]\n"
          "# $35 = dew point temperature [K]\n"
          "# $36 = frost point temperature [K]\n"
          "# $37 = NAT temperature [K]\n"
          "# $38 = HNO3 volume mixing ratio [ppv]\n"
          "# $39 = OH concentration [molec/cm^3]\n"
          "# $40 = H2O2 concentration [molec/cm^3]\n");
  fprintf(out,
          "# $41 = boundary layer pressure [hPa]\n"
          "# $42 = number of data points\n"
          "# $43 = number of tropopause data points\n"
          "# $44 = number of CAPE data points\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 %g %g %g %g %g %g %g"
              " %g %g %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
              timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
              lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
              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], tsm[ix][iy] / np[ix][iy],
              zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
              vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
              ztm[ix][iy] / npt[ix][iy], ttm[ix][iy] / npt[ix][iy],
              h2otm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
              iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
              pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
              plclm[ix][iy] / npc[ix][iy], plfcm[ix][iy] / npc[ix][iy],
              pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
              cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
              rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
              ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
```

```
                hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
                h2o2m[ix][iy] / np[ix][iy], pblm[ix][iy] / np[ix][iy],
                np[ix][iy], npt[ix][iy], npc[ix][iy]);
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

5.31 met_prof.c File Reference

Extract vertical profile from meteorological data.

```
#include "libtrac.h"
```

Macros

- #define NZ 1000

  Maximum number of altitudes.

Functions

- int main (int argc, char ∗argv[ ])

5.31.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met_prof.c.

5.31.2 Macro Definition Documentation

5.31.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line 32 of file met_prof.c.

5.31.3 Function Documentation

5.31.3.1 main()

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

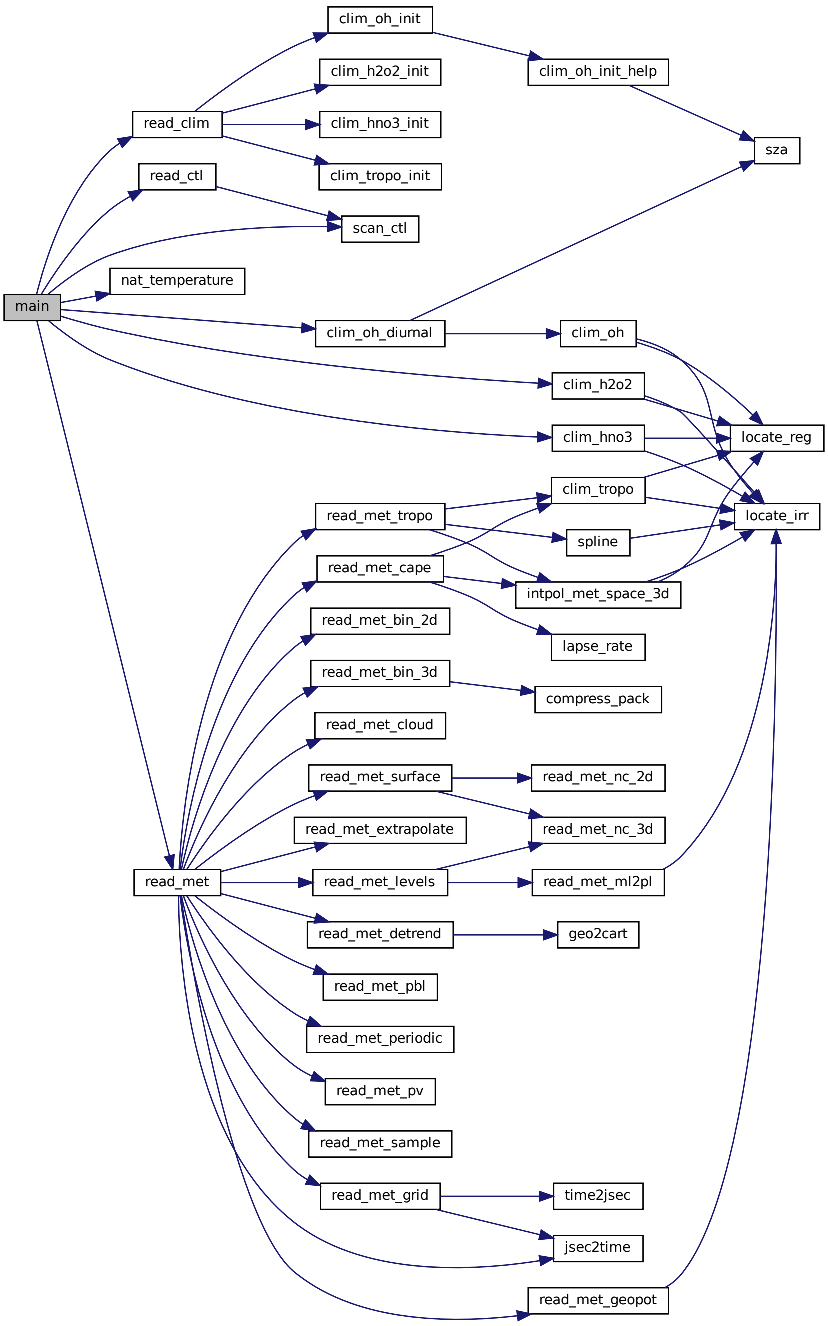
Definition at line 38 of file met_prof.c.

```
                     {

  ctl_t ctl;

  clim_t *clim;

  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], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
    vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
    cl, clm[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
    cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
    pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
    tnatm[NZ], hno3m[NZ], ohm[NZ], h2o2m[NZ], cw[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, 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_SPACE_ALL(plev[iz], lon, lat);

          /* 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;
            lwcm[iz] += lwc;
            iwcm[iz] += iwc;
            psm[iz] += ps;
            tsm[iz] += ts;
            zsm[iz] += zs;
            usm[iz] += us;
            vsm[iz] += vs;
            pblm[iz] += pbl;
            pctm[iz] += pct;
            pcbm[iz] += pcb;
            clm[iz] += cl;
            if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
                && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
              plclm[iz] += plcl;
              plfcm[iz] += plfc;
              pelm[iz] += pel;
              capem[iz] += cape;
              cinm[iz] += cin;
              npc[iz]++;
            }
            if (gsl_finite(pt)) {
              ptm[iz] += pt;
              ztm[iz] += zt;
              ttm[iz] += tt;
              h2otm[iz] += h2ot;
              npt[iz]++;
            }
            rhm[iz] += RH(plev[iz], t, h2o);
            rhicem[iz] += RHICE(plev[iz], t, h2o);
            tdewm[iz] += TDEW(plev[iz], h2o);
            ticem[iz] += TICE(plev[iz], h2o);
            hno3m[iz] += clim_hno3(clim, met->time, lat, plev[iz]);
            tnatm[iz] +=
              nat_temperature(plev[iz], h2o,
                              clim_hno3(clim, met->time, lat, plev[iz]));
            ohm[iz] +=
              clim_oh_diurnal(&ctl, clim, met->time, plev[iz], lon, lat);
            h2o2m[iz] += clim_h2o2(clim, met->time, lat, plev[iz]);
            np[iz]++;
          }
        }
  }

  /* Create output file... */
  LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
          "# $16 = surface geopotential height [km]\n"
```

```
          "# $17 = surface zonal wind [m/s]\n"
          "# $18 = surface meridional wind [m/s]\n"
          "# $19 = tropopause pressure [hPa]\n"
          "# $20 = tropopause geopotential height [km]\n");
  fprintf(out,
          "# $21 = tropopause temperature [K]\n"
          "# $22 = tropopause water vapor [ppv]\n"
          "# $23 = cloud liquid water content [kg/kg]\n"
          "# $24 = cloud ice water content [kg/kg]\n"
          "# $25 = total column cloud water [kg/m^2]\n"
          "# $26 = cloud top pressure [hPa]\n"
          "# $27 = cloud bottom pressure [hPa]\n"
          "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
          "# $29 = pressure at level of free convection (LFC) [hPa]\n"
          "# $30 = pressure at equilibrium level (EL) [hPa]\n");
  fprintf(out,
          "# $31 = convective available potential energy (CAPE) [J/kg]\n"
          "# $32 = convective inhibition (CIN) [J/kg]\n"
          "# $33 = relative humidity over water [%%]\n"
          "# $34 = relative humidity over ice [%%]\n"
          "# $35 = dew point temperature [K]\n"
          "# $36 = frost point temperature [K]\n"
          "# $37 = NAT temperature [K]\n"
          "# $38 = HNO3 volume mixing ratio [ppv]\n"
          "# $39 = OH concentration [molec/cm^3]\n"
          "# $40 = H2O2 concentration [molec/cm^3]\n");
  fprintf(out,
          "# $41 = boundary layer pressure [hPa]\n"
          "# $42 = number of data points\n"
          "# $43 = number of tropopause data points\n"
          "# $44 = number of CAPE data points\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 %g %g %g %g %g %g %g %g %g"
            " %g %g %g %g %g %g %g %g %g %g %d %d %d\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], tsm[iz] / np[iz], zsm[iz] / np[iz],
            usm[iz] / np[iz], vsm[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],
            pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
            plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
            cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
            tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
            hno3m[iz] / np[iz], ohm[iz] / np[iz], h2o2m[iz] / np[iz],
            pblm[iz] / np[iz], np[iz], npt[iz], npc[iz]);

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.32 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) 2013-2022 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

#define NZ 1000

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

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

  ctl_t ctl;

  clim_t *clim;

  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], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
    vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
    cl, clm[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
    cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
    pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
    tnatm[NZ], hno3m[NZ], ohm[NZ], h2o2m[NZ], cw[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, 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_SPACE_ALL(plev[iz], lon, lat);

        /* 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;
          lwcm[iz] += lwc;
          iwcm[iz] += iwc;
          psm[iz] += ps;
          tsm[iz] += ts;
          zsm[iz] += zs;
          usm[iz] += us;
          vsm[iz] += vs;
          pblm[iz] += pbl;
          pctm[iz] += pct;
          pcbm[iz] += pcb;
          clm[iz] += cl;
          if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
              && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
            plclm[iz] += plcl;
            plfcm[iz] += plfc;
            pelm[iz] += pel;
            capem[iz] += cape;
            cinm[iz] += cin;
            npc[iz]++;
          }
          if (gsl_finite(pt)) {
            ptm[iz] += pt;
            ztm[iz] += zt;
            ttm[iz] += tt;
            h2otm[iz] += h2ot;
            npt[iz]++;
          }
          rhm[iz] += RH(plev[iz], t, h2o);
          rhicem[iz] += RHICE(plev[iz], t, h2o);
          tdewm[iz] += TDEW(plev[iz], h2o);
          ticem[iz] += TICE(plev[iz], h2o);
          hno3m[iz] += clim_hno3(clim, met->time, lat, plev[iz]);
          tnatm[iz] +=
            nat_temperature(plev[iz], h2o,
                            clim_hno3(clim, met->time, lat, plev[iz]));
          ohm[iz] +=
            clim_oh_diurnal(&ctl, clim, met->time, plev[iz], lon, lat);
          h2o2m[iz] += clim_h2o2(clim, met->time, lat, plev[iz]);
          np[iz]++;
        }
      }
}

/* Create output file... */
LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
          "# $16 = surface geopotential height [km]\n"
          "# $17 = surface zonal wind [m/s]\n"
          "# $18 = surface meridional wind [m/s]\n"
          "# $19 = tropopause pressure [hPa]\n"
          "# $20 = tropopause geopotential height [km]\n");
  fprintf(out,
          "# $21 = tropopause temperature [K]\n"
          "# $22 = tropopause water vapor [ppv]\n"
          "# $23 = cloud liquid water content [kg/kg]\n"
          "# $24 = cloud ice water content [kg/kg]\n"
          "# $25 = total column cloud water [kg/m^2]\n"
          "# $26 = cloud top pressure [hPa]\n"
          "# $27 = cloud bottom pressure [hPa]\n"
          "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
          "# $29 = pressure at level of free convection (LFC) [hPa]\n"
          "# $30 = pressure at equilibrium level (EL) [hPa]\n");
  fprintf(out,
          "# $31 = convective available potential energy (CAPE) [J/kg]\n"
          "# $32 = convective inhibition (CIN) [J/kg]\n"
          "# $33 = relative humidity over water [%%]\n"
          "# $34 = relative humidity over ice [%%]\n"
          "# $35 = dew point temperature [K]\n"
          "# $36 = frost point temperature [K]\n"
          "# $37 = NAT temperature [K]\n"
          "# $38 = HNO3 volume mixing ratio [ppv]\n"
          "# $39 = OH concentration [molec/cm^3]\n"
          "# $40 = H2O2 concentration [molec/cm^3]\n");
  fprintf(out,
          "# $41 = boundary layer pressure [hPa]\n"
          "# $42 = number of data points\n"
          "# $43 = number of tropopause data points\n"
          "# $44 = number of CAPE data points\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 %g %g %g %g %g %g %g %g"
            " %g %g %g %g %g %g %g %g %g %g %d %d %d\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], tsm[iz] / np[iz], zsm[iz] / np[iz],
            usm[iz] / np[iz], vsm[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],
            pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
            plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
            cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
            tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
            hno3m[iz] / np[iz], ohm[iz] / np[iz], h2o2m[iz] / np[iz],
            pblm[iz] / np[iz], np[iz], npt[iz], npc[iz]);

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

5.33 met_sample.c File Reference

Sample meteorological data at given geolocations.

```
#include "libtrac.h"
```

Functions

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

5.33.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

5.33.2 Function Documentation

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

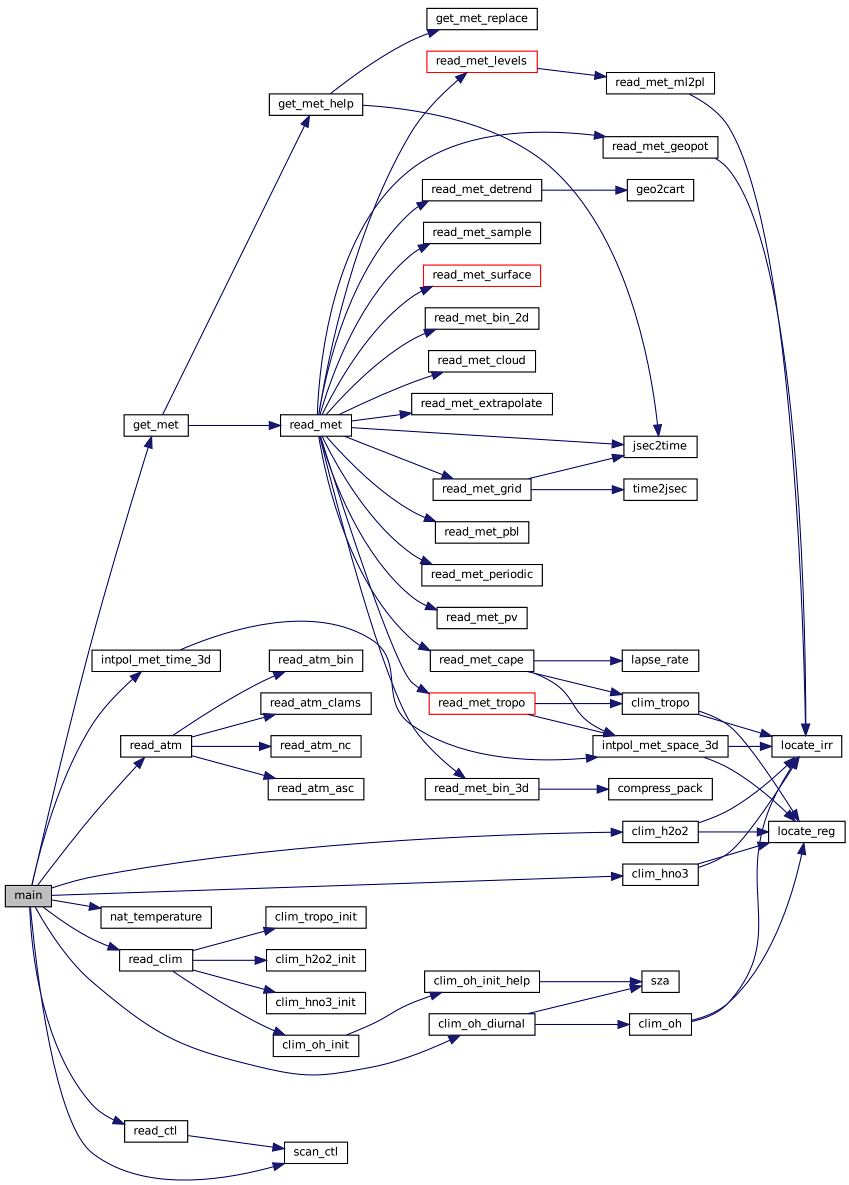
Definition at line 31 of file met_sample.c.

```
00033                  {
00034
00035   ctl_t ctl;
00036
00037   clim_t *clim;
00038
00039   atm_t *atm;
00040
00041   met_t *met0, *met1;
00042
00043   FILE *out;
00044
00045   double h2o, h2ot, o3, lwc, iwc, p0, p1, ps, ts, zs, us, vs, pbl, pt,
00046     pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00047     zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00048
00049   int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00050
00051   /* Check arguments... */
00052   if (argc < 3)
00053     ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00054
00055   /* Allocate... */
00056   ALLOC(clim, clim_t, 1);
00057   ALLOC(atm, atm_t, 1);
00058   ALLOC(met0, met_t, 1);
00059   ALLOC(met1, met_t, 1);
00060
00061   /* Read control parameters... */
00062   read_ctl(argv[1], argc, argv, &ctl);
00063   geopot =
00064     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00065   grid_time =
00066     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00067   grid_z =
00068     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00069   grid_lon =
00070     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00071   grid_lat =
00072     (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00073
00074   /* Read climatological data... */
00075   read_clim(&ctl, clim);
00076
00077   /* Read atmospheric data... */
00078   if (!read_atm(argv[3], &ctl, atm))
```

```
    ERRMSG("Cannot open file!");

  /* Create output file... */
  LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
          "# $16 = surface geopotential height [km]\n"
          "# $17 = surface zonal wind [m/s]\n"
          "# $18 = surface meridional wind [m/s]\n"
          "# $19 = tropopause pressure [hPa]\n"
          "# $20 = tropopause geopotential height [km]\n");
  fprintf(out,
          "# $21 = tropopause temperature [K]\n"
          "# $22 = tropopause water vapor [ppv]\n"
          "# $23 = cloud liquid water content [kg/kg]\n"
          "# $24 = cloud ice water content [kg/kg]\n"
          "# $25 = total column cloud water [kg/m^2]\n"
          "# $26 = cloud top pressure [hPa]\n"
          "# $27 = cloud bottom pressure [hPa]\n"
          "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
          "# $29 = pressure at level of free convection (LFC) [hPa]\n"
          "# $30 = pressure at equilibrium level (EL) [hPa]\n");
  fprintf(out,
          "# $31 = convective available potential energy (CAPE) [J/kg]\n"
          "# $32 = convective inhibition (CIN) [J/kg]\n"
          "# $33 = relative humidity over water [%%]\n"
          "# $34 = relative humidity over ice [%%]\n"
          "# $35 = dew point temperature [K]\n"
          "# $36 = frost point temperature [K]\n"
          "# $37 = NAT temperature [K]\n"
          "# $38 = HNO3 volume mixing ratio [ppv]\n"
          "# $39 = OH concentration [molec/cm^3]\n"
          "# $40 = H2O2 concentration [molec/cm^3]\n"
          "# $41 = boundary layer pressure [hPa]\n");

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

    /* Get meteorological data... */
    get_met(&ctl, clim, atm->time[ip], &met0, &met1);

    /* Set reference pressure for interpolation... */
    double 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_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);

    /* Make blank lines... */
    if (ip == 0 || (grid_time && atm->time[ip] != time_old)
        || (grid_z && atm->p[ip] != p_old)
        || (grid_lon && atm->lon[ip] != lon_old)
        || (grid_lat && atm->lat[ip] != lat_old))
      fprintf(out, "\n");
```

```
    time_old = atm->time[ip];
    p_old = atm->p[ip];
    lon_old = atm->lon[ip];
    lat_old = atm->lat[ip];

    /* Write data... */
    fprintf(out,
            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g"
            " %g %g %g %g %g %g %g %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, ts, zs, us, vs,
            pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
            cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
            TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
            nat_temperature(atm->p[ip], h2o,
                            clim_hno3(clim, atm->time[ip], atm->lat[ip],
                                      atm->p[ip])), clim_hno3(clim,
                                                              atm->time[ip],
                                                              atm->lat[ip],
                                                              atm->p[ip]),
            clim_oh_diurnal(&ctl, clim, atm->time[ip], atm->p[ip],
                            atm->lon[ip], atm->lat[ip]),
            clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
  }

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

  /* Free... */
  free(clim);
  free(atm);
  free(met0);
  free(met1);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.34 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) 2013-2022 Forschungszentrum Juelich GmbH
*/
```

```
#include "libtrac.h"

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

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

  ctl_t ctl;

  clim_t *clim;

  atm_t *atm;

  met_t *met0, *met1;

  FILE *out;

  double h2o, h2ot, o3, lwc, iwc, p0, p1, ps, ts, zs, us, vs, pbl, pt,
    pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
    zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;

  int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  grid_time =
    (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
  grid_z =
    (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
  grid_lon =
    (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
  grid_lat =
    (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

  /* Create output file... */
  LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
          "# $16 = surface geopotential height [km]\n"
          "# $17 = surface zonal wind [m/s]\n"
          "# $18 = surface meridional wind [m/s]\n"
          "# $19 = tropopause pressure [hPa]\n"
          "# $20 = tropopause geopotential height [km]\n");
  fprintf(out,
          "# $21 = tropopause temperature [K]\n"
```

```
          "# $22 = tropopause water vapor [ppv]\n"
          "# $23 = cloud liquid water content [kg/kg]\n"
          "# $24 = cloud ice water content [kg/kg]\n"
          "# $25 = total column cloud water [kg/m^2]\n"
          "# $26 = cloud top pressure [hPa]\n"
          "# $27 = cloud bottom pressure [hPa]\n"
          "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
          "# $29 = pressure at level of free convection (LFC) [hPa]\n"
          "# $30 = pressure at equilibrium level (EL) [hPa]\n");
  fprintf(out,
          "# $31 = convective available potential energy (CAPE) [J/kg]\n"
          "# $32 = convective inhibition (CIN) [J/kg]\n"
          "# $33 = relative humidity over water [%%]\n"
          "# $34 = relative humidity over ice [%%]\n"
          "# $35 = dew point temperature [K]\n"
          "# $36 = frost point temperature [K]\n"
          "# $37 = NAT temperature [K]\n"
          "# $38 = HNO3 volume mixing ratio [ppv]\n"
          "# $39 = OH concentration [molec/cm^3]\n"
          "# $40 = H2O2 concentration [molec/cm^3]\n"
          "# $41 = boundary layer pressure [hPa]\n");

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

    /* Get meteorological data... */
    get_met(&ctl, clim, atm->time[ip], &met0, &met1);

    /* Set reference pressure for interpolation... */
    double 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_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);

    /* Make blank lines... */
    if (ip == 0 || (grid_time && atm->time[ip] != time_old)
        || (grid_z && atm->p[ip] != p_old)
        || (grid_lon && atm->lon[ip] != lon_old)
        || (grid_lat && atm->lat[ip] != lat_old))
      fprintf(out, "\n");
    time_old = atm->time[ip];
    p_old = atm->p[ip];
    lon_old = atm->lon[ip];
    lat_old = atm->lat[ip];

    /* Write data... */
    fprintf(out,
            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g"
            " %g %g %g %g %g %g %g %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, ts, zs, us, vs,
            pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
            cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
            TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
            nat_temperature(atm->p[ip], h2o,
                            clim_hno3(clim, atm->time[ip], atm->lat[ip],
                                      atm->p[ip])), clim_hno3(clim,
                                                              atm->time[ip],
                                                              atm->lat[ip],
                                                              atm->p[ip]),
            clim_oh_diurnal(&ctl, clim, atm->time[ip], atm->p[ip],
                            atm->lon[ip], atm->lat[ip]),
            clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
  }

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

  /* Free... */
  free(clim);
  free(atm);
  free(met0);
```

```
00198    free(met1);
00199
00200    return EXIT_SUCCESS;
00201 }
```

5.35 met_spec.c File Reference

Spectral analysis of meteorological data.

`#include "libtrac.h"`

Macros

- #define PMAX EX

  Maximum number of data points for spectral analysis.

Functions

- void fft_help (double ∗fcReal, double ∗fcImag, int n)
- int main (int argc, char ∗argv[ ])

5.35.1 Detailed Description

Spectral analysis of meteorological data.

Definition in file met_spec.c.

5.35.2 Macro Definition Documentation

5.35.2.1 PMAX `#define PMAX EX`

Maximum number of data points for spectral analysis.

Definition at line 32 of file met_spec.c.

5.35.3 Function Documentation

5.35.3.1 fft_help() void fft_help (
double * *fcReal*,
double * *fcImag*,
int *n*)

Definition at line 150 of file met_spec.c.

```
                {

  gsl_fft_complex_wavetable *wavetable;
  gsl_fft_complex_workspace *workspace;

  double data[2 * PMAX];

  int i;

  /* Check size... */
  if (n > PMAX)
    ERRMSG("Too many data points!");

  /* Allocate... */
  wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
  workspace = gsl_fft_complex_workspace_alloc((size_t) n);

  /* Set data (real, complex)... */
  for (i = 0; i < n; i++) {
    data[2 * i] = fcReal[i];
    data[2 * i + 1] = fcImag[i];
  }

  /* Calculate FFT... */
  gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);

  /* Copy data... */
  for (i = 0; i < n; i++) {
    fcReal[i] = data[2 * i];
    fcImag[i] = data[2 * i + 1];
  }

  /* Free... */
  gsl_fft_complex_wavetable_free(wavetable);
  gsl_fft_complex_workspace_free(workspace);
}
```

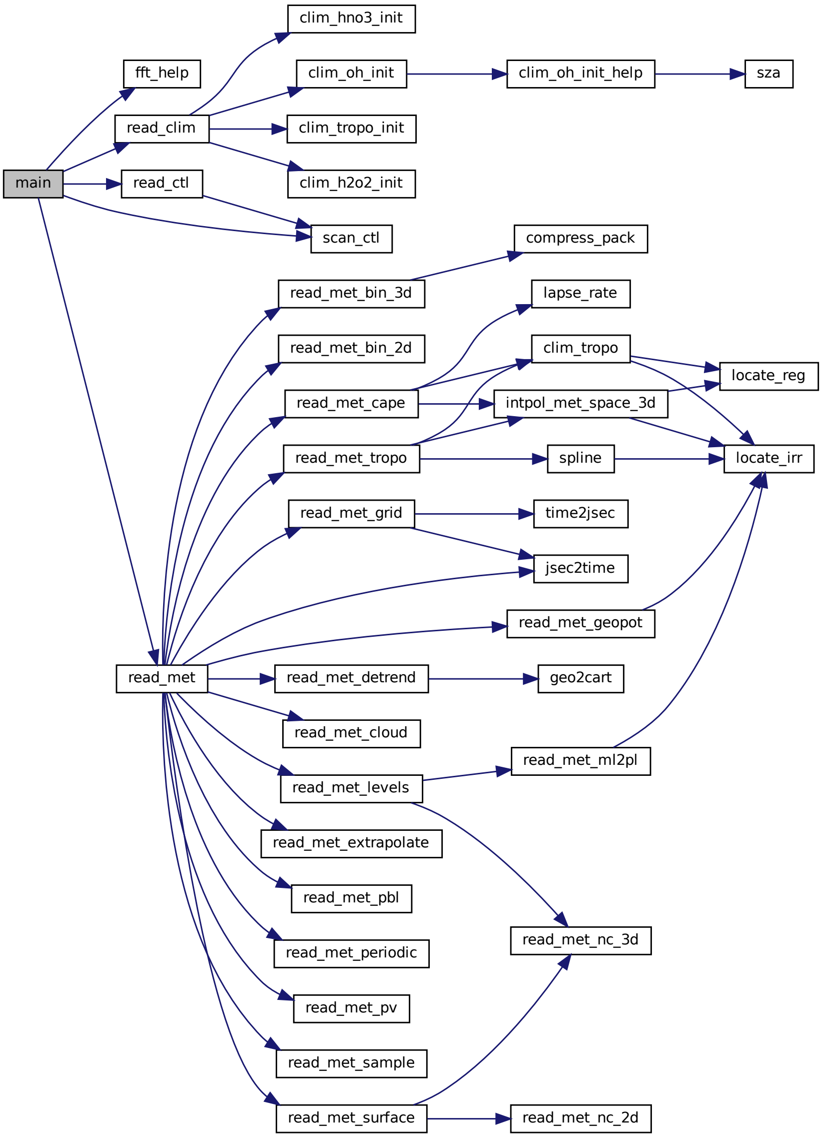
5.35.3.2 main() int main (
int *argc*,
char * *argv[ ]*)

Definition at line 47 of file met_spec.c.

```
                       {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
    wavemax;

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  wavemax =
    (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

```

```
  /* Read meteorological data... */
  if (!read_met(argv[3], &ctl, clim, met))
    ERRMSG("Cannot read meteo data!");

  /* Create output file... */
  LOG(1, "Write spectral data file: %s", 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 (int ix = 0; ix <= wavemax; ix++) {
    fprintf(out, "# $%d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
    fprintf(out, "# $%d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
    fprintf(out, "# $%d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
  }

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

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

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

      /* Copy data... */
      for (int ix = 0; ix < met->nx; ix++) {
        cutReal[ix] = met->t[ix][iy][ip];
        cutImag[ix] = 0.0;
      }

      /* FFT... */
      fft_help(cutReal, cutImag, met->nx);

      /*
         Get wavelength, amplitude, and phase:
         A(x) = A[0] + A[1] * cos(2 pi x / lx[1] + phi[1]) + A[2] * cos...
       */
      for (int ix = 0; ix < met->nx; ix++) {
        lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
          / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
        A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
          * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
        phi[ix]
          = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
      }

      /* Write data... */
      fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
              met->lat[iy]);
      for (int ix = 0; ix <= wavemax; ix++)
        fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
      fprintf(out, "\n");
    }
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.36 met_spec.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) 2013-2021 Forschungszentrum Juelich GmbH
*/

```

```
#include "libtrac.h"

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

#define PMAX EX

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

void fft_help(
  double *fcReal,
  double *fcImag,
  int n);

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

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
    wavemax;

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met, met_t, 1);

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  wavemax =
    (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

  /* Read meteorological data... */
  if (!read_met(argv[3], &ctl, clim, met))
    ERRMSG("Cannot read meteo data!");

  /* Create output file... */
  LOG(1, "Write spectral data file: %s", 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 (int ix = 0; ix <= wavemax; ix++) {
    fprintf(out, "# $%d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
    fprintf(out, "# $%d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
    fprintf(out, "# $%d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
  }

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

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

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

      /* Copy data... */
      for (int ix = 0; ix < met->nx; ix++) {
        cutReal[ix] = met->t[ix][iy][ip];
        cutImag[ix] = 0.0;
      }

```

```
      /* FFT... */
      fft_help(cutReal, cutImag, met->nx);

      /*
         Get wavelength, amplitude, and phase:
         A(x) = A[0] + A[1] * cos(2 pi x / lx[1] + phi[1]) + A[2] * cos...
       */
      for (int ix = 0; ix < met->nx; ix++) {
        lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
          / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
        A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
          * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
        phi[ix]
          = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
      }

      /* Write data... */
      fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
              met->lat[iy]);
      for (int ix = 0; ix <= wavemax; ix++)
        fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
      fprintf(out, "\n");
    }
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}

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

void fft_help(
  double *fcReal,
  double *fcImag,
  int n) {

  gsl_fft_complex_wavetable *wavetable;
  gsl_fft_complex_workspace *workspace;

  double data[2 * PMAX];

  int i;

  /* Check size... */
  if (n > PMAX)
    ERRMSG("Too many data points!");

  /* Allocate... */
  wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
  workspace = gsl_fft_complex_workspace_alloc((size_t) n);

  /* Set data (real, complex)... */
  for (i = 0; i < n; i++) {
    data[2 * i] = fcReal[i];
    data[2 * i + 1] = fcImag[i];
  }

  /* Calculate FFT... */
  gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);

  /* Copy data... */
  for (i = 0; i < n; i++) {
    fcReal[i] = data[2 * i];
    fcImag[i] = data[2 * i + 1];
  }

  /* Free... */
  gsl_fft_complex_wavetable_free(wavetable);
  gsl_fft_complex_workspace_free(workspace);
}
```

5.37 met_subgrid.c File Reference

Calculate standard deviations of horizontal wind and vertical velocity.

```
#include "libtrac.h"
```

Functions

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

5.37.1 Detailed Description

Calculate standard deviations of horizontal wind and vertical velocity.

Definition in file met_subgrid.c.

5.37.2 Function Documentation

5.37.2.1 main()

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

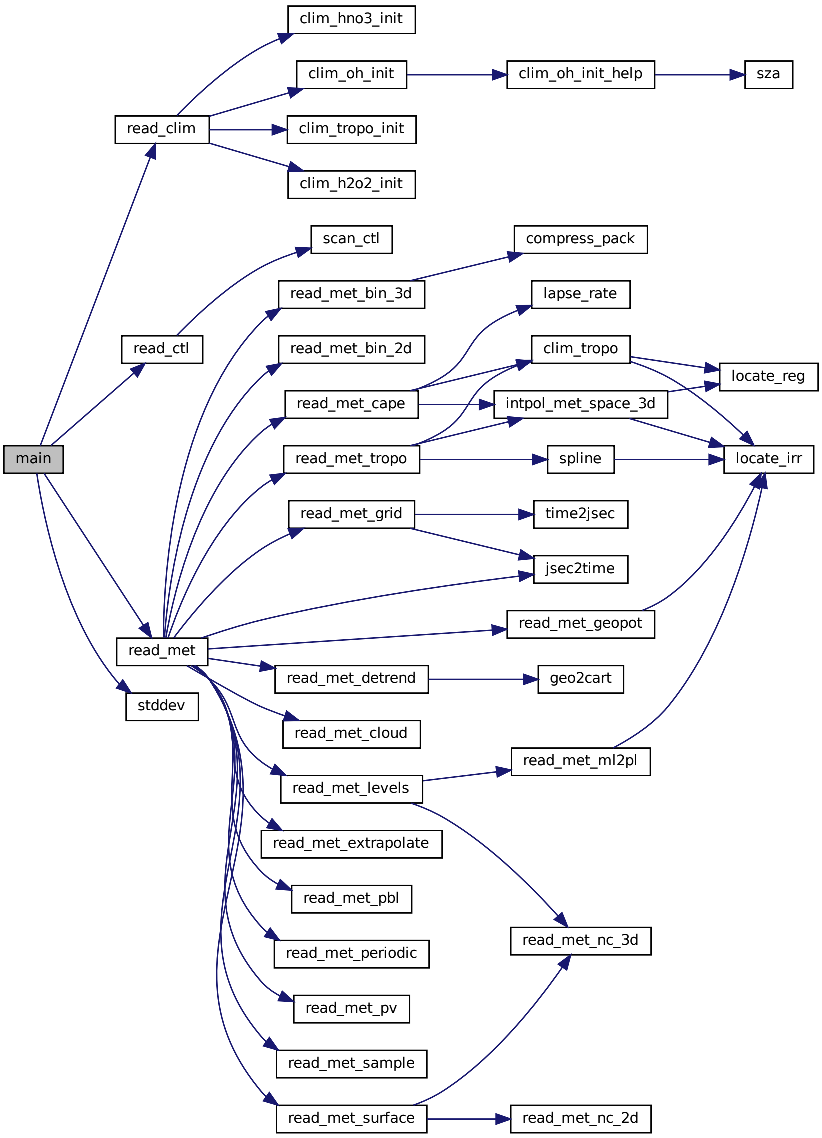
Definition at line 31 of file met_subgrid.c.

```
                  {

  ctl_t ctl;

  clim_t *clim;

  met_t *met0, *met1;

  FILE *out;

  static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];

  static float u[16], v[16], w[16];

  static int i, ix, iy, iz, n[EP][EY];

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);

  /* Check arguments... */
  if (argc < 4 && argc % 2 != 0)
    ERRMSG
      ("Give parameters: <ctl> <subgrid.tab> <met0> <met1> [ <met0> <met1> ... ]");

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

  /* Read climatological data... */
  read_clim(&ctl, clim);

  /* Loop over data files... */
  for (i = 3; i < argc - 1; i += 2) {

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, met0))
      ERRMSG("Cannot open file!");
    if (!read_met(argv[i + 1], &ctl, clim, met1))
      ERRMSG("Cannot open file!");

    /* Loop over grid boxes... */
    for (ix = 0; ix < met0->nx - 1; ix++)
      for (iy = 0; iy < met0->ny - 1; iy++)
        for (iz = 0; iz < met0->np - 1; iz++) {

          /* 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] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
          w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
          w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
          w[3] =
            (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
          w[4] =
            (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
          w[5] =
            (float) (1e3 *
                     DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
          w[6] =
            (float) (1e3 *
                     DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
          w[7] =
            (float) (1e3 *
                     DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[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] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
          w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
          w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
          w[11] =
            (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
          w[12] =
            (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
          w[13] =
            (float) (1e3 *
                     DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
          w[14] =
            (float) (1e3 *
                     DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
          w[15] =
            (float) (1e3 *
                     DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));

          /* Get standard deviations of local wind data... */
          usig[iz][iy] += stddev(u, 16);
          vsig[iz][iy] += stddev(v, 16);
          wsig[iz][iy] += stddev(w, 16);
          n[iz][iy]++;

          /* Check surface pressure... */
          if (met0->p[iz] > met0->ps[ix][iy]
              || met1->p[iz] > met1->ps[ix][iy]) {
            usig[iz][iy] = GSL_NAN;
            vsig[iz][iy] = GSL_NAN;
            wsig[iz][iy] = GSL_NAN;
            n[iz][iy] = 0;
          }
        }
  }

  /* Create output file... */
  LOG(1, "Write subgrid data file: %s", 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 = zonal wind standard deviation [m/s]\n"
          "# $6 = meridional wind standard deviation [m/s]\n"
          "# $7 = vertical velocity standard deviation [m/s]\n"
          "# $8 = number of data points\n");

  /* Write output... */
  for (iy = 0; iy < met0->ny - 1; iy++) {
    fprintf(out, "\n");
    for (iz = 0; iz < met0->np - 1; iz++)
      fprintf(out, "%.2f %g %g %g %g %g %g %d\n",
              0.5 * (met0->time + met1->time),
              0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
              0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
              usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
              wsig[iz][iy] / n[iz][iy], n[iz][iy]);
  }

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

  /* Free... */
  free(clim);
  free(met0);
  free(met1);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.38 met_subgrid.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) 2013-2021 Forschungszentrum Juelich GmbH
*/

```

```c
#include "libtrac.h"

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

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met0, *met1;

  FILE *out;

  static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];

  static float u[16], v[16], w[16];

  static int i, ix, iy, iz, n[EP][EY];

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);

  /* Check arguments... */
  if (argc < 4 && argc % 2 != 0)
    ERRMSG
      ("Give parameters: <ctl> <subgrid.tab> <met0> <met1> [ <met0> <met1> ... ]");

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

  /* Read climatological data... */
  read_clim(&ctl, clim);

  /* Loop over data files... */
  for (i = 3; i < argc - 1; i += 2) {

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, met0))
      ERRMSG("Cannot open file!");
    if (!read_met(argv[i + 1], &ctl, clim, met1))
      ERRMSG("Cannot open file!");

    /* Loop over grid boxes... */
    for (ix = 0; ix < met0->nx - 1; ix++)
      for (iy = 0; iy < met0->ny - 1; iy++)
        for (iz = 0; iz < met0->np - 1; iz++) {

          /* 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] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
          w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
          w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
          w[3] =
            (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
          w[4] =
            (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
          w[5] =
            (float) (1e3 *
                     DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
          w[6] =
            (float) (1e3 *
                     DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
          w[7] =
```

```
            (float) (1e3 *
                     DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[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] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
          w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
          w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
          w[11] =
            (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
          w[12] =
            (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
          w[13] =
            (float) (1e3 *
                     DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
          w[14] =
            (float) (1e3 *
                     DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
          w[15] =
            (float) (1e3 *
                     DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));

          /* Get standard deviations of local wind data... */
          usig[iz][iy] += stddev(u, 16);
          vsig[iz][iy] += stddev(v, 16);
          wsig[iz][iy] += stddev(w, 16);
          n[iz][iy]++;

          /* Check surface pressure... */
          if (met0->p[iz] > met0->ps[ix][iy]
              || met1->p[iz] > met1->ps[ix][iy]) {
            usig[iz][iy] = GSL_NAN;
            vsig[iz][iy] = GSL_NAN;
            wsig[iz][iy] = GSL_NAN;
            n[iz][iy] = 0;
          }
        }
  }

  /* Create output file... */
  LOG(1, "Write subgrid data file: %s", 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 = zonal wind standard deviation [m/s]\n"
          "# $6 = meridional wind standard deviation [m/s]\n"
          "# $7 = vertical velocity standard deviation [m/s]\n"
          "# $8 = number of data points\n");

  /* Write output... */
  for (iy = 0; iy < met0->ny - 1; iy++) {
    fprintf(out, "\n");
    for (iz = 0; iz < met0->np - 1; iz++)
      fprintf(out, "%.2f %g %g %g %g %g %g %d\n",
              0.5 * (met0->time + met1->time),
              0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
              0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
              usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
              wsig[iz][iy] / n[iz][iy], n[iz][iy]);
  }

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

```

```
    /* Free... */
    free(clim);
    free(met0);
    free(met1);

    return EXIT_SUCCESS;
}
```

5.39 met_zm.c File Reference

Extract zonal mean from meteorological data.

```
#include "libtrac.h"
```

Macros

- #define NZ 1000

  Maximum number of altitudes.
- #define NY 721

  Maximum number of latitudes.

Functions

- int main (int argc, char ∗argv[ ])

5.39.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met_zm.c.

5.39.2 Macro Definition Documentation

5.39.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line 32 of file met_zm.c.

5.39.2.2 NY `#define NY 721`

Maximum number of latitudes.

Definition at line 35 of file met_zm.c.

5.39.3 Function Documentation

5.39.3.1 main()

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

Definition at line 41 of file met_zm.c.

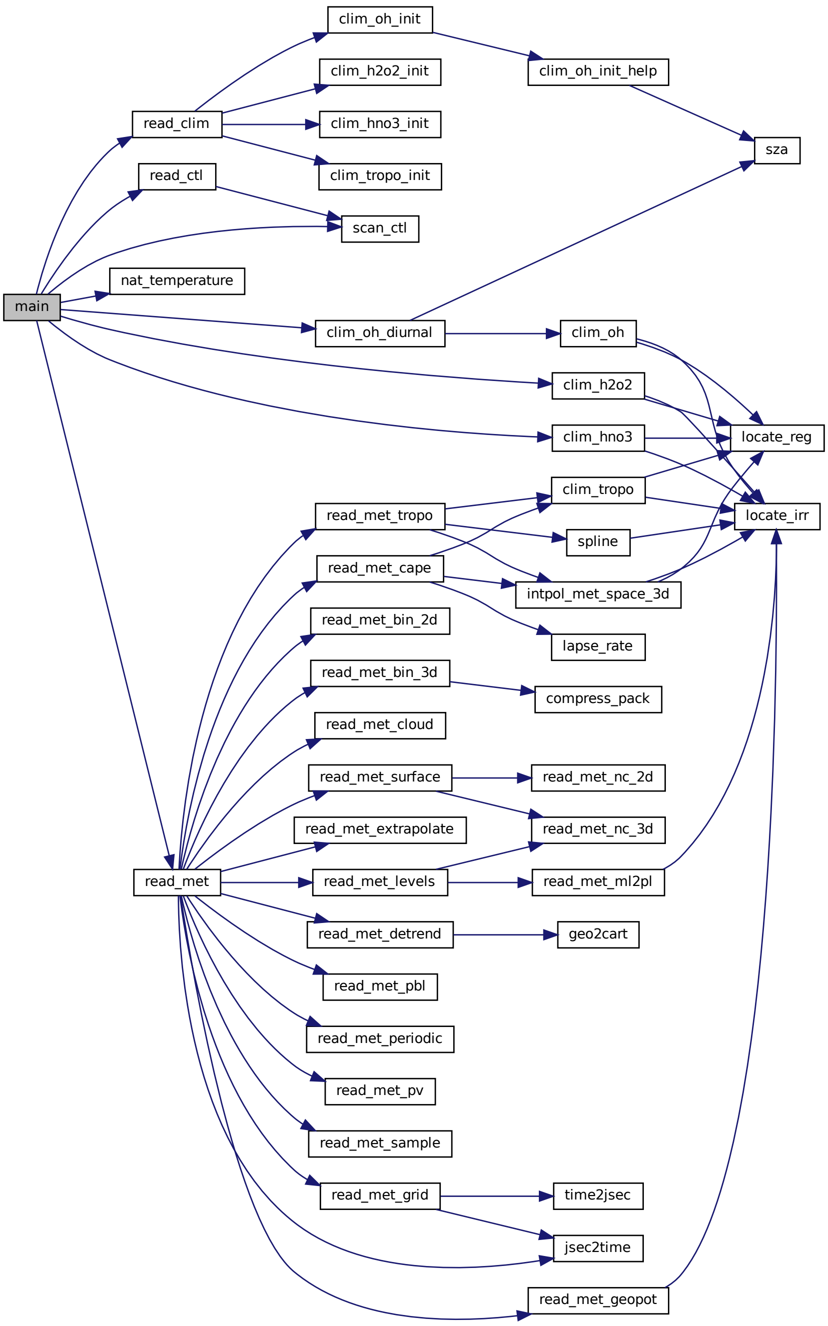
```
                    {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
    usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
    pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
    capem[NZ][NY], cinm[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],
    rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
    hno3m[NZ][NY], ohm[NZ][NY], h2o2m[NZ][NY], z, z0, z1, dz, zt, tt,
    plev[NZ], ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
    cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
    lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", 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);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, 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++)
      if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
        for (iy = 0; iy < ny; iy++)
          for (iz = 0; iz < nz; iz++) {

            /* Interpolate meteo data... */
            INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);

            /* Averaging... */
            timem[iz][iy] += met->time;
            lonm[iz][iy] += met->lon[ix];
            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;
            tsm[iz][iy] += ts;
            zsm[iz][iy] += zs;
            usm[iz][iy] += us;
            vsm[iz][iy] += vs;
            pblm[iz][iy] += pbl;
            pctm[iz][iy] += pct;
            pcbm[iz][iy] += pcb;
            clm[iz][iy] += cl;
            if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
                && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
              plclm[iz][iy] += plcl;
              plfcm[iz][iy] += plfc;
              pelm[iz][iy] += pel;
              capem[iz][iy] += cape;
              cinm[iz][iy] += cin;
              npc[iz][iy]++;
            }
            if (gsl_finite(pt)) {
              ptm[iz][iy] += pt;
              ztm[iz][iy] += zt;
              ttm[iz][iy] += tt;
              h2otm[iz][iy] += h2ot;
              npt[iz][iy]++;
            }
            rhm[iz][iy] += RH(plev[iz], t, h2o);
            rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
            tdewm[iz][iy] += TDEW(plev[iz], h2o);
            ticem[iz][iy] += TICE(plev[iz], h2o);
            hno3m[iz][iy] += clim_hno3(clim, met->time, lats[iy], plev[iz]);
            tnatm[iz][iy] +=
              nat_temperature(plev[iz], h2o,
                              clim_hno3(clim, met->time, lats[iy], plev[iz]));
            ohm[iz][iy] +=
              clim_oh_diurnal(&ctl, clim, met->time, plev[iz], met->lon[ix],
                              lats[iy]);
            h2o2m[iz][iy] += clim_h2o2(clim, met->time, lats[iy], plev[iz]);
            np[iz][iy]++;
          }
  }

  /* Create output file... */
  LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
        "# $16 = surface geopotential height [km]\n"
        "# $17 = surface zonal wind [m/s]\n"
        "# $18 = surface meridional wind [m/s]\n"
        "# $19 = tropopause pressure [hPa]\n"
        "# $20 = tropopause geopotential height [km]\n");
fprintf(out,
        "# $21 = tropopause temperature [K]\n"
        "# $22 = tropopause water vapor [ppv]\n"
        "# $23 = cloud liquid water content [kg/kg]\n"
        "# $24 = cloud ice water content [kg/kg]\n"
        "# $25 = total column cloud water [kg/m^2]\n"
        "# $26 = cloud top pressure [hPa]\n"
        "# $27 = cloud bottom pressure [hPa]\n"
        "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
        "# $29 = pressure at level of free convection (LFC) [hPa]\n"
        "# $30 = pressure at equilibrium level (EL) [hPa]\n");
fprintf(out,
        "# $31 = convective available potential energy (CAPE) [J/kg]\n"
        "# $32 = convective inhibition (CIN) [J/kg]\n"
        "# $33 = relative humidity over water [%%]\n"
        "# $34 = relative humidity over ice [%%]\n"
        "# $35 = dew point temperature [K]\n"
        "# $36 = frost point temperature [K]\n"
        "# $37 = NAT temperature [K]\n"
        "# $38 = HNO3 volume mixing ratio [ppv]\n"
        "# $39 = OH concentration [molec/cm^3]\n"
        "# $40 = H2O2 concentration [molec/cm^3]\n");
fprintf(out,
        "# $41 = boundary layer pressure [hPa]\n"
        "# $42 = number of data points\n"
        "# $43 = number of tropopause data points\n"
        "# $44 = number of CAPE data points\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 %g %g %g %g %g %g %g %g %g %g %g %g %g %g"
            " %g %g %g %g %g %d %d %d\n",
            timem[iz][iy] / np[iz][iy], Z(plev[iz]),
            lonm[iz][iy] / np[iz][iy], 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], tsm[iz][iy] / np[iz][iy],
            zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
            vsm[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],
            pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],
            plclm[iz][iy] / npc[iz][iy], plfcm[iz][iy] / npc[iz][iy],
            pelm[iz][iy] / npc[iz][iy], capem[iz][iy] / npc[iz][iy],
            cinm[iz][iy] / npc[iz][iy], rhm[iz][iy] / np[iz][iy],
            rhicem[iz][iy] / np[iz][iy], tdewm[iz][iy] / np[iz][iy],
            ticem[iz][iy] / np[iz][iy], tnatm[iz][iy] / np[iz][iy],
            hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
            h2o2m[iz][iy] / np[iz][iy], pblm[iz][iy] / np[iz][iy],
            np[iz][iy], npt[iz][iy], npc[iz][iy]);
}

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

/* Free... */
free(clim);
free(met);

return EXIT_SUCCESS;
```

```
00289 }
```

Here is the call graph for this function:



5.40 met_zm.c

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

```
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) 2013-2022 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

#define NZ 1000

#define NY 721

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

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  FILE *out;

  static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
    usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
    pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
    capem[NZ][NY], cinm[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],
    rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
    hno3m[NZ][NY], ohm[NZ][NY], h2o2m[NZ][NY], z, z0, z1, dz, zt, tt,
    plev[NZ], ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
    cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
    lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];

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

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
  lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", 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);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    if (!read_met(argv[i], &ctl, clim, 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++)
    if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
      for (iy = 0; iy < ny; iy++)
        for (iz = 0; iz < nz; iz++) {

          /* Interpolate meteo data... */
          INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);

          /* Averaging... */
          timem[iz][iy] += met->time;
          lonm[iz][iy] += met->lon[ix];
          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;
          tsm[iz][iy] += ts;
          zsm[iz][iy] += zs;
          usm[iz][iy] += us;
          vsm[iz][iy] += vs;
          pblm[iz][iy] += pbl;
          pctm[iz][iy] += pct;
          pcbm[iz][iy] += pcb;
          clm[iz][iy] += cl;
          if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
              && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
            plclm[iz][iy] += plcl;
            plfcm[iz][iy] += plfc;
            pelm[iz][iy] += pel;
            capem[iz][iy] += cape;
            cinm[iz][iy] += cin;
            npc[iz][iy]++;
          }
          if (gsl_finite(pt)) {
            ptm[iz][iy] += pt;
            ztm[iz][iy] += zt;
            ttm[iz][iy] += tt;
            h2otm[iz][iy] += h2ot;
            npt[iz][iy]++;
          }
          rhm[iz][iy] += RH(plev[iz], t, h2o);
          rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
          tdewm[iz][iy] += TDEW(plev[iz], h2o);
          ticem[iz][iy] += TICE(plev[iz], h2o);
          hno3m[iz][iy] += clim_hno3(clim, met->time, lats[iy], plev[iz]);
          tnatm[iz][iy] +=
            nat_temperature(plev[iz], h2o,
                            clim_hno3(clim, met->time, lats[iy], plev[iz]));
          ohm[iz][iy] +=
            clim_oh_diurnal(&ctl, clim, met->time, plev[iz], met->lon[ix],
                            lats[iy]);
```

```
          h2o2m[iz][iy] += clim_h2o2(clim, met->time, lats[iy], plev[iz]);
          np[iz][iy]++;
        }
  }

  /* Create output file... */
  LOG(1, "Write meteorological data file: %s", 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 velocity [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 = surface temperature [K]\n"
          "# $16 = surface geopotential height [km]\n"
          "# $17 = surface zonal wind [m/s]\n"
          "# $18 = surface meridional wind [m/s]\n"
          "# $19 = tropopause pressure [hPa]\n"
          "# $20 = tropopause geopotential height [km]\n");
  fprintf(out,
          "# $21 = tropopause temperature [K]\n"
          "# $22 = tropopause water vapor [ppv]\n"
          "# $23 = cloud liquid water content [kg/kg]\n"
          "# $24 = cloud ice water content [kg/kg]\n"
          "# $25 = total column cloud water [kg/m^2]\n"
          "# $26 = cloud top pressure [hPa]\n"
          "# $27 = cloud bottom pressure [hPa]\n"
          "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
          "# $29 = pressure at level of free convection (LFC) [hPa]\n"
          "# $30 = pressure at equilibrium level (EL) [hPa]\n");
  fprintf(out,
          "# $31 = convective available potential energy (CAPE) [J/kg]\n"
          "# $32 = convective inhibition (CIN) [J/kg]\n"
          "# $33 = relative humidity over water [%%]\n"
          "# $34 = relative humidity over ice [%%]\n"
          "# $35 = dew point temperature [K]\n"
          "# $36 = frost point temperature [K]\n"
          "# $37 = NAT temperature [K]\n"
          "# $38 = HNO3 volume mixing ratio [ppv]\n"
          "# $39 = OH concentration [molec/cm^3]\n"
          "# $40 = H2O2 concentration [molec/cm^3]\n");
  fprintf(out,
          "# $41 = boundary layer pressure [hPa]\n"
          "# $42 = number of data points\n"
          "# $43 = number of tropopause data points\n"
          "# $44 = number of CAPE data points\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 %g %g %g %g %g %g %g %g %g %g %g %g %g %g"
              " %g %g %g %g %g %d %d %d\n",
              timem[iz][iy] / np[iz][iy], Z(plev[iz]),
              lonm[iz][iy] / np[iz][iy], 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], tsm[iz][iy] / np[iz][iy],
              zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
              vsm[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],
              pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],
              plclm[iz][iy] / npc[iz][iy], plfcm[iz][iy] / npc[iz][iy],
              pelm[iz][iy] / npc[iz][iy], capem[iz][iy] / npc[iz][iy],
              cinm[iz][iy] / npc[iz][iy], rhm[iz][iy] / np[iz][iy],
              rhicem[iz][iy] / np[iz][iy], tdewm[iz][iy] / np[iz][iy],
              ticem[iz][iy] / np[iz][iy], tnatm[iz][iy] / np[iz][iy],
```

```
              hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
              h2o2m[iz][iy] / np[iz][iy], pblm[iz][iy] / np[iz][iy],
              np[iz][iy], npt[iz][iy], npc[iz][iy]);
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

5.41 sedi.c File Reference

Calculate sedimentation velocity.

```
#include "libtrac.h"
```

Functions

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

5.41.1 Detailed Description

Calculate sedimentation velocity.

Definition in file sedi.c.

5.41.2 Function Documentation

5.41.2.1 main()

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

Definition at line 27 of file sedi.c.

```
                {

  double eta, p, T, r_p, rho, Re, rho_p, vs;

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <p> <T> <r_p> <rho_p>");

  /* Read arguments... */
  p = atof(argv[1]);
  T = atof(argv[2]);
  r_p = atof(argv[3]);
  rho_p = atof(argv[4]);

  /* Calculate sedimentation velocity... */
  vs = sedi(p, T, r_p, rho_p);

  /* Density of dry air [kg / m^3]... */
  rho = 100. * p / (RA * T);

  /* Dynamic viscosity of air [kg / (m s)]... */
```

```c
  eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Particle Reynolds number... */
  Re = 2e-6 * r_p * vs * rho / eta;

  /* Write output... */
  printf("     p= %g hPa\n", p);
  printf("     T= %g K\n", T);
  printf("   r_p= %g microns\n", r_p);
  printf("rho_p= %g kg/m^3\n", rho_p);
  printf("rho_a= %g kg/m^3\n", RHO(p, T));
  printf("   v_s= %g m/s\n", vs);
  printf("    Re= %g\n", Re);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.42 sedi.c

```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) 2013-2021 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

  double eta, p, T, r_p, rho, Re, rho_p, vs;

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <p> <T> <r_p> <rho_p>");

  /* Read arguments... */
  p = atof(argv[1]);
  T = atof(argv[2]);
  r_p = atof(argv[3]);
  rho_p = atof(argv[4]);

  /* Calculate sedimentation velocity... */
  vs = sedi(p, T, r_p, rho_p);

  /* Density of dry air [kg / m^3]... */
  rho = 100. * p / (RA * T);

  /* Dynamic viscosity of air [kg / (m s)]... */
  eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);

  /* Particle Reynolds number... */
```

```
  Re = 2e-6 * r_p * vs * rho / eta;

  /* Write output... */
  printf("    p= %g hPa\n", p);
  printf("    T= %g K\n", T);
  printf("  r_p= %g microns\n", r_p);
  printf("rho_p= %g kg/m^3\n", rho_p);
  printf("rho_a= %g kg/m^3\n", RHO(p, T));
  printf("  v_s= %g m/s\n", vs);
  printf("   Re= %g\n", Re);

  return EXIT_SUCCESS;
}
```

5.43 time2jsec.c File Reference

Convert date to Julian seconds.

```
#include "libtrac.h"
```

Functions

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

5.43.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

5.43.2 Function Documentation

5.43.2.1 main()

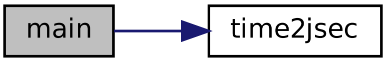
```
int main (
    int argc,
    char * argv[ ] )
```

Definition at line 27 of file time2jsec.c.

```
                  {

  double jsec, remain;

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

  /* Check arguments... */
  if (argc < 8)
    ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");

  /* Read arguments... */
  year = atoi(argv[1]);
  mon = atoi(argv[2]);
  day = atoi(argv[3]);
  hour = atoi(argv[4]);
  min = atoi(argv[5]);
  sec = atoi(argv[6]);
  remain = atof(argv[7]);

  /* Convert... */
  time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
  printf("%.2f\n", jsec);
```

```
  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.44 time2jsec.c

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

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

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

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

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

#include "libtrac.h"

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

  double jsec, remain;

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

  /* Check arguments... */
  if (argc < 8)
    ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");

  /* Read arguments... */
  year = atoi(argv[1]);
  mon = atoi(argv[2]);
  day = atoi(argv[3]);
  hour = atoi(argv[4]);
  min = atoi(argv[5]);
  sec = atoi(argv[6]);
  remain = atof(argv[7]);

  /* Convert... */
  time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
  printf("%.2f\n", jsec);

  return EXIT_SUCCESS;
}
```

5.45 tnat.c File Reference

Calculate PSC temperatures.

```
#include "libtrac.h"
```

Functions

- int main (int argc, char ∗argv[ ])

5.45.1 Detailed Description

Calculate PSC temperatures.

Definition in file tnat.c.

5.45.2 Function Documentation

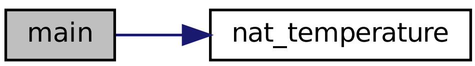
5.45.2.1 main()

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

Definition at line 31 of file tnat.c.

```
                    {

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <p> <h2o> <hno3>");

  /* Get varibles... */
  double p = atof(argv[1]);
  double h2o = atof(argv[2]);
  double hno3 = atof(argv[3]);

  /* Write output... */
  printf("      p= %g hPa\n", p);
  printf(" q_H2O= %g ppv\n", h2o);
  printf("q_HNO3= %g ppv\n", hno3);
  printf(" T_dew= %g K\n", TDEW(p, h2o));
  printf(" T_ice= %g K\n", TICE(p, h2o));
  printf(" T_NAT= %g K\n", nat_temperature(p, h2o, hno3));

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.46 tnat.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) 2013-2023 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

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

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <p> <h2o> <hno3>");

  /* Get varibles... */
  double p = atof(argv[1]);
  double h2o = atof(argv[2]);
  double hno3 = atof(argv[3]);

  /* Write output... */
  printf("      p= %g hPa\n", p);
  printf(" q_H2O= %g ppv\n", h2o);
  printf("q_HNO3= %g ppv\n", hno3);
  printf(" T_dew= %g K\n", TDEW(p, h2o));
  printf(" T_ice= %g K\n", TICE(p, h2o));
  printf(" T_NAT= %g K\n", nat_temperature(p, h2o, hno3));

  return EXIT_SUCCESS;
}
```

5.47 trac.c File Reference

Lagrangian particle dispersion model.

```
#include "libtrac.h"
```

Functions

- void module_advect (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double *dt)

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

  Apply boundary conditions.
- void module_convection (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double *dt, double *rs)

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

  Calculate exponential decay of particle mass.
- 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_turb (ctl_t ∗ctl, clim_t ∗clim, atm_t ∗atm, double ∗dt, double ∗rs)

  Calculate turbulent diffusion.
- void module_dry_deposition (ctl_t ∗ctl, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double ∗dt)

  Calculate dry deposition.
- 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, clim_t ∗clim, met_t ∗met0, met_t ∗met1, atm_t ∗atm)

  Interpolate meteo data for air parcel positions.
- void module_oh_chem (ctl_t ∗ctl, clim_t ∗clim, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double ∗dt)

  Calculate OH chemistry.
- void module_h2o2_chem (ctl_t ∗ctl, clim_t ∗clim, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double ∗dt, double ∗rs)

  Calculate H2O2 chemistry.
- void module_chemgrid (ctl_t ∗ctl, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double t)

  Interpolate to chemistry grid.
- void module_position (ctl_t ∗ctl, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double ∗dt)

  Check position of air parcels.
- void module_rng_init (int ntask)

  Initialize random number generator...
- void module_rng (double ∗rs, size_t n, int method)

  Generate random numbers.
- void module_sedi (ctl_t ∗ctl, met_t ∗met0, met_t ∗met1, atm_t ∗atm, double ∗dt)

  Calculate sedimentation of air parcels.
- void module_sort (ctl_t ∗ctl, met_t ∗met0, atm_t ∗atm)

  Sort particles according to box index.
- void module_sort_help (double ∗a, int ∗p, int np)

  Helper function for sorting module.
- void module_timesteps (ctl_t ∗ctl, atm_t ∗atm, met_t ∗met0, double ∗dt, double t)

  Calculate time steps.
- void module_timesteps_init (ctl_t ∗ctl, atm_t ∗atm)

  Initialize timesteps.
- 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[ ])

5.47.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.47.2 Function Documentation

5.47.2.1 module_advect()

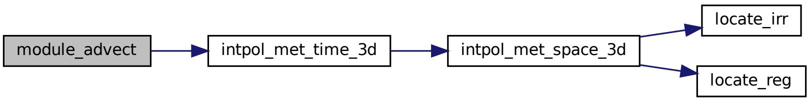
```
void module_advect (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )
```

Calculate advection of air parcels.

Definition at line 558 of file trac.c.

```
                     {

  /* Set timer... */
  SELECT_TIMER("MODULE_ADVECTION", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Init... */
      double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];

      /* Loop over integration nodes... */
      for (int i = 0; i < ctl->advect; i++) {

        /* Set position... */
        if (i == 0) {
          dts = 0.0;
          x[0] = atm->lon[ip];
          x[1] = atm->lat[ip];
          x[2] = atm->p[ip];
        } else {
          dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
          x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
          x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
          x[2] = atm->p[ip] + dts * w[i - 1];
        }
        double tm = atm->time[ip] + dts;

        /* Interpolate meteo data... */
#ifdef UVW
        intpol_met_time_uvw(met0, met1, tm, x[2], x[0], x[1],
                            &u[i], &v[i], &w[i]);
#else
        INTPOL_INIT;
        intpol_met_time_3d(met0, met0->u, met1, met1->u, tm,
                           x[2], x[0], x[1], &u[i], ci, cw, 1);
        intpol_met_time_3d(met0, met0->v, met1, met1->v, tm,
                           x[2], x[0], x[1], &v[i], ci, cw, 0);
        intpol_met_time_3d(met0, met0->w, met1, met1->w, tm,
                           x[2], x[0], x[1], &w[i], ci, cw, 0);
#endif

        /* Get mean wind... */
        double k = 1.0;
        if (ctl->advect == 2)
          k = (i == 0 ? 0.0 : 1.0);
        else if (ctl->advect == 4)
          k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
        um += k * u[i];
        vm += k * v[i];
        wm += k * w[i];
      }

      /* Set new position... */
      atm->time[ip] += dt[ip];
      atm->lon[ip] += DX2DEG(dt[ip] * um / 1000.,
                             (ctl->advect == 2 ? x[1] : atm->lat[ip]));
      atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
      atm->p[ip] += dt[ip] * wm;
    }
}
```

Here is the call graph for this function:



5.47.2.2 module_bound_cond()

```
void module_bound_cond (
        ctl_t * ctl,
        met_t * met0,
        met_t * met1,
        atm_t * atm,
        double * dt )
```

Apply boundary conditions.

Definition at line 634 of file trac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Check latitude and pressure range... */
      if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
          || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
        continue;

      /* Check surface layer... */
      if (ctl->bound_dps > 0 || ctl->bound_dzs > 0
          || ctl->bound_zetas > 0 || ctl->bound_pbl) {

        /* Get surface pressure... */
        double ps;
        INTPOL_INIT;
        INTPOL_2D(ps, 1);

        /* Check pressure... */
        if (ctl->bound_dps > 0 && atm->p[ip] < ps - ctl->bound_dps)
          continue;

        /* Check height... */
        if (ctl->bound_dzs > 0 && Z(atm->p[ip]) > Z(ps) + ctl->bound_dzs)
          continue;

        /* Check zeta range... */
        if (ctl->bound_zetas > 0) {
          double t;
          INTPOL_3D(t, 1);
          if (ZETA(ps, atm->p[ip], t) > ctl->bound_zetas)
            continue;
        }

        /* Check planetary boundary layer... */
```

```
        if (ctl->bound_pbl) {
          double pbl;
          INTPOL_2D(pbl, 0);
          if (atm->p[ip] < pbl)
            continue;
        }
      }

      /* Set mass and volume mixing ratio... */
      if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
        atm->q[ctl->qnt_m][ip] =
          ctl->bound_mass + ctl->bound_mass_trend * atm->time[ip];
      if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] =
          ctl->bound_vmr + ctl->bound_vmr_trend * atm->time[ip];
    }
}
```

5.47.2.3 module_convection()

```
void module_convection (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt,
    double * rs )
```

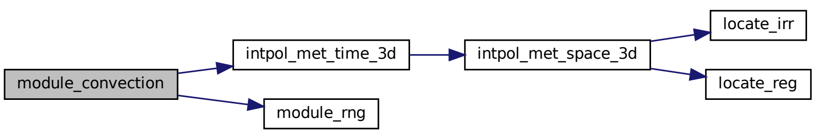
Calculate convection of air parcels.

Definition at line 709 of file trac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);

  /* Create random numbers... */
  module_rng(rs, (size_t) atm->np, 0);

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

      double cape, cin, pel, ps;

      /* Interpolate CAPE... */
      INTPOL_INIT;
      INTPOL_2D(cape, 1);

      /* Check threshold... */
      if (isfinite(cape) && cape >= ctl->conv_cape) {

        /* Check CIN... */
        if (ctl->conv_cin > 0) {
          INTPOL_2D(cin, 0);
          if (isfinite(cin) && cin >= ctl->conv_cin)
            continue;
        }

        /* Interpolate equilibrium level... */
        INTPOL_2D(pel, 0);

        /* Check whether particle is above cloud top... */
        if (!isfinite(pel) || atm->p[ip] < pel)
          continue;

        /* Set pressure range for mixing... */
        double pbot = atm->p[ip];
        double ptop = atm->p[ip];
        if (ctl->conv_mix_bot == 1) {
          INTPOL_2D(ps, 0);
          pbot = ps;
```

```
        }
        if (ctl->conv_mix_top == 1)
          ptop = pel;

        /* Vertical mixing based on pressure... */
        if (ctl->conv_mix == 0)
          atm->p[ip] = pbot + (ptop - pbot) * rs[ip];

        /* Vertical mixing based on density... */
        else if (ctl->conv_mix == 1) {

          /* Get density range... */
          double tbot, ttop;
          intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                             pbot, atm->lon[ip], atm->lat[ip], &tbot,
                             ci, cw, 1);
          intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                             ptop, atm->lon[ip], atm->lat[ip], &ttop,
                             ci, cw, 1);
          double rhobot = pbot / tbot;
          double rhotop = ptop / ttop;

          /* Get new density... */
          double lrho = log(rhobot + (rhotop - rhobot) * rs[ip]);

          /* Find pressure... */
          double lrhobot = log(rhobot);
          double lrhotop = log(rhotop);
          double lpbot = log(pbot);
          double lptop = log(ptop);
          atm->p[ip] = exp(LIN(lrhobot, lpbot, lrhotop, lptop, lrho));
        }
      }
    }
}
```

Here is the call graph for this function:



5.47.2.4 module_decay() `void module_decay ( ctl_t * ctl, clim_t * clim, atm_t * atm, double * dt )`

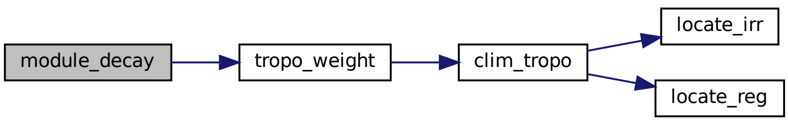
Calculate exponential decay of particle mass.

Definition at line 800 of file trac.c.

```
              {

  /* Set timer... */
  SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#ifdef _OPENACC
#pragma acc data present(ctl,clim,atm,dt)
#pragma acc parallel loop independent gang vector
#else
```

```
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < np; ip++)
    if (dt[ip] != 0) {

      /* Get weighting factor... */
      double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);

      /* Set lifetime... */
      double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;

      /* Calculate exponential decay... */
      double aux = exp(-dt[ip] / tdec);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_decay >= 0)
          atm->q[ctl->qnt_mloss_decay][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}
```

Here is the call graph for this function:



5.47.2.5 module_diffusion_meso()

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

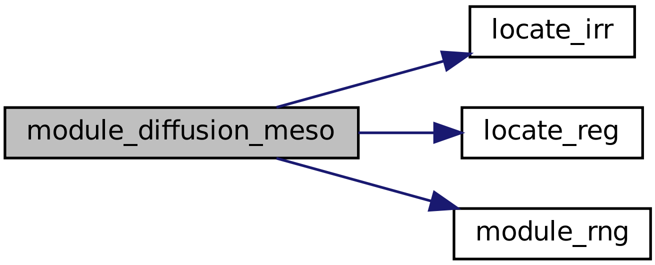
Calculate mesoscale diffusion.

Definition at line 844 of file trac.c.

```
                   {

  /* Set timer... */
  SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);

  /* Create random numbers... */
  module_rng(rs, 3 * (size_t) atm->np, 1);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* 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]);

      /* Get standard deviations of local wind data... */
      float umean = 0, usig = 0, vmean = 0, vsig = 0, wmean = 0, wsig = 0;
      for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
          for (int k = 0; k < 2; k++) {
#ifdef UVW
            umean += met0->uvw[ix + i][iy + j][iz + k][0];
            usig += SQR(met0->uvw[ix + i][iy + j][iz + k][0]);
            vmean += met0->uvw[ix + i][iy + j][iz + k][1];
            vsig += SQR(met0->uvw[ix + i][iy + j][iz + k][1]);
            wmean += met0->uvw[ix + i][iy + j][iz + k][2];
            wsig += SQR(met0->uvw[ix + i][iy + j][iz + k][2]);

            umean += met1->uvw[ix + i][iy + j][iz + k][0];
            usig += SQR(met1->uvw[ix + i][iy + j][iz + k][0]);
            vmean += met1->uvw[ix + i][iy + j][iz + k][1];
            vsig += SQR(met1->uvw[ix + i][iy + j][iz + k][1]);
            wmean += met1->uvw[ix + i][iy + j][iz + k][2];
            wsig += SQR(met1->uvw[ix + i][iy + j][iz + k][2]);
#else
            umean += met0->u[ix + i][iy + j][iz + k];
            usig += SQR(met0->u[ix + i][iy + j][iz + k]);
            vmean += met0->v[ix + i][iy + j][iz + k];
            vsig += SQR(met0->v[ix + i][iy + j][iz + k]);
            wmean += met0->w[ix + i][iy + j][iz + k];
            wsig += SQR(met0->w[ix + i][iy + j][iz + k]);

            umean += met1->u[ix + i][iy + j][iz + k];
            usig += SQR(met1->u[ix + i][iy + j][iz + k]);
            vmean += met1->v[ix + i][iy + j][iz + k];
            vsig += SQR(met1->v[ix + i][iy + j][iz + k]);
            wmean += met1->w[ix + i][iy + j][iz + k];
            wsig += SQR(met1->w[ix + i][iy + j][iz + k]);
#endif
          }
      usig = usig / 16.f - SQR(umean / 16.f);
      usig = (usig > 0 ? sqrtf(usig) : 0);
      vsig = vsig / 16.f - SQR(vmean / 16.f);
      vsig = (vsig > 0 ? sqrtf(vsig) : 0);
      wsig = wsig / 16.f - SQR(wmean / 16.f);
      wsig = (wsig > 0 ? sqrtf(wsig) : 0);

      /* 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->uvwp[ip][0] =
          (float) (r * cache->uvwp[ip][0] +
                   r2 * rs[3 * ip] * ctl->turb_mesox * usig);
        atm->lon[ip] +=
          DX2DEG(cache->uvwp[ip][0] * dt[ip] / 1000., atm->lat[ip]);

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

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

Here is the call graph for this function:



5.47.2.6 module_diffusion_turb()

```
void module_diffusion_turb (
    ctl_t * ctl,
    clim_t * clim,
    atm_t * atm,
    double * dt,
    double * rs )
```

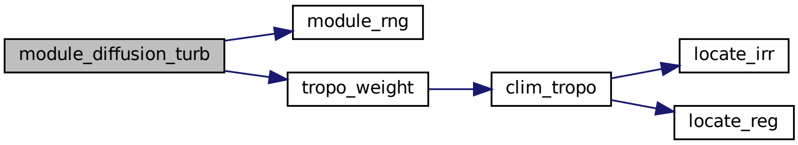
Calculate turbulent diffusion.

Definition at line 946 of file trac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);

  /* Create random numbers... */
  module_rng(rs, 3 * (size_t) atm->np, 1);

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

      /* Get weighting factor... */
      double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], 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:



5.47.2.7 module_dry_deposition()

```
void module_dry_deposition (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )
```

Calculate dry deposition.

Definition at line 993 of file trac.c.

```
                {

  /* Set timer... */
  SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      double ps, t, v_dep;

      /* Get surface pressure... */
      INTPOL_INIT;
      INTPOL_2D(ps, 1);

      /* Check whether particle is above the surface layer... */
      if (atm->p[ip] < ps - ctl->dry_depo_dp)
        continue;

      /* Set depth of surface layer... */
      double dz = 1000. * (Z(ps - ctl->dry_depo_dp) - Z(ps));

      /* Calculate sedimentation velocity for particles... */
      if (ctl->qnt_rp > 0 && ctl->qnt_rhop > 0) {

        /* Get temperature... */
        INTPOL_3D(t, 1);

        /* Set deposition velocity... */
        v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
                     atm->q[ctl->qnt_rhop][ip]);
      }

      /* Use explicit sedimentation velocity for gases... */
      else
```

```
        v_dep = ctl->dry_depo_vdep;

      /* Calculate loss of mass based on deposition velocity... */
      double aux = exp(-dt[ip] * v_dep / dz);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_dry >= 0)
          atm->q[ctl->qnt_mloss_dry][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}
```

Here is the call graph for this function:

5.47.2.8 module_isosurf_init()

```
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 1060 of file trac.c.

```
                   {

  FILE *in;

  char line[LEN];

  double t;

  /* Set timer... */
  SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);

  /* Init... */
  INTPOL_INIT;

  /* 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_3D(t, 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_3D(t, 1);
      cache->iso_var[ip] = THETA(atm->p[ip], t);
    }
```

```

  /* Read balloon pressure data... */
  else if (ctl->isosurf == 4) {

    /* Write info... */
    LOG(1, "Read balloon pressure data: %s", 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);
  }
}
```

5.47.2.9 module_isosurf()

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

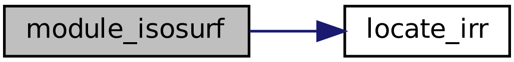
Force air parcels to stay on isosurface.

Definition at line 1126 of file trac.c.

```
                         {

  /* Set timer... */
  SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++) {

    double t;

    /* Init... */
    INTPOL_INIT;

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

    /* Restore density... */
    else if (ctl->isosurf == 2) {
      INTPOL_3D(t, 1);
      atm->p[ip] = cache->iso_var[ip] * t;
    }

    /* Restore potential temperature... */
    else if (ctl->isosurf == 3) {
      INTPOL_3D(t, 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:



5.47.2.10 module_meteo()

```
void module_meteo (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm )
```

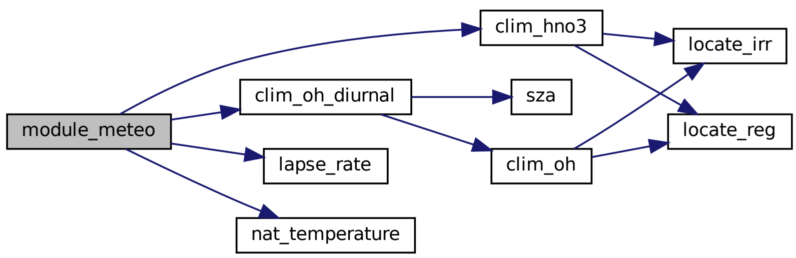
Interpolate meteo data for air parcel positions.

Definition at line 1184 of file trac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);

  /* 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!");

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

    double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
      cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;

    /* Interpolate meteo data... */
    INTPOL_INIT;
    INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);

    /* Set quantities... */
    SET_ATM(qnt_ps, ps);
    SET_ATM(qnt_ts, ts);
    SET_ATM(qnt_zs, zs);
    SET_ATM(qnt_us, us);
    SET_ATM(qnt_vs, vs);
    SET_ATM(qnt_pbl, pbl);
    SET_ATM(qnt_pt, pt);
    SET_ATM(qnt_tt, tt);
    SET_ATM(qnt_zt, zt);
```

```
    SET_ATM(qnt_h2ot, h2ot);
    SET_ATM(qnt_z, z);
    SET_ATM(qnt_p, atm->p[ip]);
    SET_ATM(qnt_t, t);
    SET_ATM(qnt_rho, RHO(atm->p[ip], t));
    SET_ATM(qnt_u, u);
    SET_ATM(qnt_v, v);
    SET_ATM(qnt_w, w);
    SET_ATM(qnt_h2o, h2o);
    SET_ATM(qnt_o3, o3);
    SET_ATM(qnt_lwc, lwc);
    SET_ATM(qnt_iwc, iwc);
    SET_ATM(qnt_pct, pct);
    SET_ATM(qnt_pcb, pcb);
    SET_ATM(qnt_cl, cl);
    SET_ATM(qnt_plcl, plcl);
    SET_ATM(qnt_plfc, plfc);
    SET_ATM(qnt_pel, pel);
    SET_ATM(qnt_cape, cape);
    SET_ATM(qnt_cin, cin);
    SET_ATM(qnt_hno3,
            clim_hno3(clim, atm->time[ip], atm->lat[ip], atm->p[ip]));
    SET_ATM(qnt_oh,
            clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
                            atm->lon[ip], atm->lat[ip]));
    SET_ATM(qnt_vh, sqrt(u * u + v * v));
    SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
    SET_ATM(qnt_psat, PSAT(t));
    SET_ATM(qnt_psice, PSICE(t));
    SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
    SET_ATM(qnt_sh, SH(h2o));
    SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
    SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
    SET_ATM(qnt_theta, THETA(atm->p[ip], t));
    SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
    SET_ATM(qnt_tvirt, TVIRT(t, h2o));
    SET_ATM(qnt_lapse, lapse_rate(t, h2o));
    SET_ATM(qnt_pv, pv);
    SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
    SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
    SET_ATM(qnt_tnat,
            nat_temperature(atm->p[ip], h2o,
                            clim_hno3(clim, atm->time[ip], atm->lat[ip],
                                      atm->p[ip])));
    SET_ATM(qnt_tsts,
            0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
  }
}
```

Here is the call graph for this function:



5.47.2.11 module_oh_chem()

```
void module_oh_chem (
        ctl_t * ctl,
        clim_t * clim,
        met_t * met0,
        met_t * met1,
```

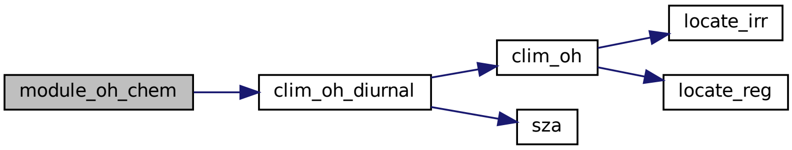
```
atm_t * atm,
double * dt )
```

Calculate OH chemistry.

Definition at line 1276 of file trac.c.

```
              {

  /* Set timer... */
  SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

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

      /* Get temperature... */
      double t;
      INTPOL_INIT;
      INTPOL_3D(t, 1);

      /* Use constant reaction rate... */
      double k = GSL_NAN;
      if (ctl->oh_chem_reaction == 1)
        k = ctl->oh_chem[0];

      /* Calculate bimolecular reaction rate... */
      else if (ctl->oh_chem_reaction == 2)
        k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);

      /* Calculate termolecular reaction rate... */
      if (ctl->oh_chem_reaction == 3) {

        /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
        double M = 7.243e21 * (atm->p[ip] / 1000.) / t;

        /* Calculate rate coefficient for X + OH + M -> XOH + M
           (JPL Publication 19-05) ... */
        double k0 = ctl->oh_chem[0] *
          (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
        double ki = ctl->oh_chem[2] *
          (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
        double c = log10(k0 * M / ki);
        k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
      }

      /* Calculate exponential decay... */
      double rate_coef =
        k * clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
                            atm->lon[ip],
                            atm->lat[ip]);
      double aux = exp(-dt[ip] * rate_coef);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_oh >= 0)
          atm->q[ctl->qnt_mloss_oh][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}
```

Here is the call graph for this function:



5.47.2.12 module_h2o2_chem()

```
void module_h2o2_chem (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt,
    double * rs )
```

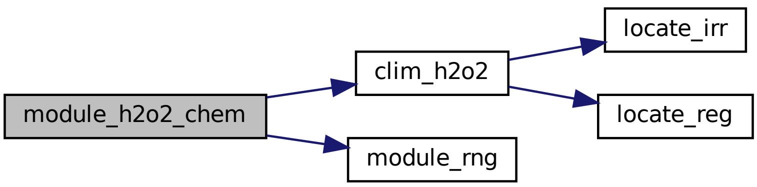
Calculate H2O2 chemistry.

Definition at line 1350 of file trac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("MODULE_H2O2CHEM", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");
  if (ctl->qnt_vmrimpl < 0)
    ERRMSG("Module needs quantity implicit volume mixing ratio!");

  /* Create random numbers... */
  module_rng(rs, (size_t) atm->np, 0);

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

      /* Check whether particle is inside cloud... */
      double lwc;
      INTPOL_INIT;
      INTPOL_3D(lwc, 1);
      if (!(lwc > 0))
        continue;

      /* Check cloud cover... */
      if (rs[ip] > ctl->h2o2_chem_cc)
        continue;

      /* Check implicit volume mixing ratio... */
      if (atm->q[ctl->qnt_vmrimpl][ip] == 0)
        continue;

      /* Get temperature... */
      double t;
      INTPOL_3D(t, 0);

      /* Reaction rate (Berglen et al., 2004)... */
```

```
      double k = 9.1e7 * exp(-29700 / RI * (1. / t - 1. / 298.15));   // Maass  1999 unit: M^(-2)

      /* Henry constant of SO2... */
      double H_SO2 = 1.3e-2 * exp(2900 * (1. / t - 1. / 298.15)) * RI * t;
      double K_1S = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));     // unit: M

      /* Henry constant of H2O2... */
      double H_h2o2 = 8.3e2 * exp(7600 * (1 / t - 1 / 298.15)) * RI * t;

      /* Concentration of H2O2 (Barth et al., 1989)... */
      double SO2 = atm->q[ctl->qnt_vmrimpl][ip] * 1e9;      // vmr unit: ppbv
      double h2o2 = H_h2o2
        * clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip])
        * 0.59 * exp(-0.687 * SO2) * 1000 / 6.02214e23; // unit: M

      /* Volume water content in cloud [m^3 m^(-3)]... */
      double rho_air = 100 * atm->p[ip] / (RA * t);
      double CWC = lwc * rho_air / 1000;

      /* Calculate exponential decay (Rolph et al., 1992)... */
      double rate_coef = k * K_1S * h2o2 * H_SO2 * CWC;
      double aux = exp(-dt[ip] * rate_coef);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_h2o2 >= 0)
          atm->q[ctl->qnt_mloss_h2o2][ip] +=
            atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}
```

Here is the call graph for this function:



5.47.2.13 module_chemgrid()

```
void module_chemgrid (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )
```

Interpolate to chemistry grid.

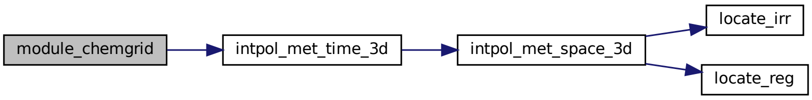
Definition at line 1436 of file trac.c.

```
                {

  double *mass, *z, *lon, *lat, *press, *area;

  int *ixs, *iys, *izs;

  /* Update host... */
#ifdef _OPENACC
  SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
#pragma acc update host(atm[:1])
```

```
#endif

  /* Set timer... */
  SELECT_TIMER("MODULE_CHEMGRID", "PHYSICS", NVTX_GPU);

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

  /* Allocate... */
  ALLOC(mass, double,
        ctl->chemgrid_nx * ctl->chemgrid_ny * ctl->chemgrid_nz);
  ALLOC(z, double,
        ctl->chemgrid_nz);
  ALLOC(lon, double,
        ctl->chemgrid_nx);
  ALLOC(lat, double,
        ctl->chemgrid_ny);
  ALLOC(area, double,
        ctl->chemgrid_ny);
  ALLOC(press, double,
        ctl->chemgrid_nz);
  ALLOC(ixs, int,
        atm->np);
  ALLOC(iys, int,
        atm->np);
  ALLOC(izs, int,
        atm->np);

  /* Set grid box size... */
  double dz = (ctl->chemgrid_z1 - ctl->chemgrid_z0) / ctl->chemgrid_nz;
  double dlon = (ctl->chemgrid_lon1 - ctl->chemgrid_lon0) / ctl->chemgrid_nx;
  double dlat = (ctl->chemgrid_lat1 - ctl->chemgrid_lat0) / ctl->chemgrid_ny;

  /* Set vertical coordinates... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < ctl->chemgrid_nz; iz++) {
    z[iz] = ctl->chemgrid_z0 + dz * (iz + 0.5);
    press[iz] = P(z[iz]);
  }

  /* Set horizontal coordinates... */
  for (int ix = 0; ix < ctl->chemgrid_nx; ix++)
    lon[ix] = ctl->chemgrid_lon0 + dlon * (ix + 0.5);
#pragma omp parallel for default(shared)
  for (int iy = 0; iy < ctl->chemgrid_ny; iy++) {
    lat[iy] = ctl->chemgrid_lat0 + dlat * (iy + 0.5);
    area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
      * cos(lat[iy] * M_PI / 180.);
  }

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

  /* Get indices... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++) {
    ixs[ip] = (int) ((atm->lon[ip] - ctl->chemgrid_lon0) / dlon);
    iys[ip] = (int) ((atm->lat[ip] - ctl->chemgrid_lat0) / dlat);
    izs[ip] = (int) ((Z(atm->p[ip]) - ctl->chemgrid_z0) / dz);
    if (atm->time[ip] < t0 || atm->time[ip] > t1
        || ixs[ip] < 0 || ixs[ip] >= ctl->chemgrid_nx
        || iys[ip] < 0 || iys[ip] >= ctl->chemgrid_ny
        || izs[ip] < 0 || izs[ip] >= ctl->chemgrid_nz)
      izs[ip] = -1;
  }

  /* Average data... */
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0)
      mass[ARRAY_3D
           (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
        += atm->q[ctl->qnt_m][ip];

  /* Interpolate volume mixing ratio... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0) {
      double temp;
      INTPOL_INIT;
      intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
                         lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
      atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass *
        mass[ARRAY_3D
```

```
              (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
          / (RHO(press[izs[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
    }

  /* Free... */
  free(mass);
  free(z);
  free(lon);
  free(lat);
  free(area);
  free(press);
  free(ixs);
  free(iys);
  free(izs);

  /* Update device... */
#ifdef _OPENACC
  SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
#pragma acc update device(atm[:1])
#endif
}
```

Here is the call graph for this function:



5.47.2.14 module_position()

```
void module_position (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )
```

Check position of air parcels.

Definition at line 1562 of file trac.c.

```
                 {

  /* Set timer... */
  SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Init... */
      double ps;
      INTPOL_INIT;

      /* 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]) {
        if (ctl->reflect)
          atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
        else
          atm->p[ip] = met0->p[met0->np - 1];
      } else if (atm->p[ip] > 300.) {
        INTPOL_2D(ps, 1);
        if (atm->p[ip] > ps) {
          if (ctl->reflect)
            atm->p[ip] = 2. * ps - atm->p[ip];
          else
            atm->p[ip] = ps;
        }
      }
    }
}
```

5.47.2.15 module_rng_init() `void module_rng_init ( int ntask )`

Initialize random number generator...

Definition at line 1628 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 (curandSetPseudoRandomGeneratorSeed(rng, ntask) != CURAND_STATUS_SUCCESS)
    ERRMSG("Cannot set seed for 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) (ntask * NTHREADS +
                                                        i));
  }

#endif
}
```

5.47.2.16 module_rng() `void module_rng ( double * rs, size_t n, int method )`

Generate random numbers.

Definition at line 1660 of file trac.c.

```
                  {

#ifdef _OPENACC

#pragma acc host_data use_device(rs)
  {
    /* Uniform distribution... */
    if (method == 0) {
      if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n)) !=
          CURAND_STATUS_SUCCESS)
        ERRMSG("Cannot create random numbers!");
    }

    /* Normal distribution... */
    else if (method == 1) {
      if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0) !=
          CURAND_STATUS_SUCCESS)
        ERRMSG("Cannot create random numbers!");
    }
  }

#else

  /* Uniform distribution... */
  if (method == 0) {
#pragma omp parallel for default(shared)
    for (size_t i = 0; i < n; ++i)
      rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
  }

  /* Normal distribution... */
  else if (method == 1) {
#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
}
```

5.47.2.17 module_sedi()

```
void module_sedi (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )
```

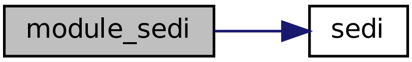
Calculate sedimentation of air parcels.

Definition at line 1704 of file trac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Get temperature... */
      double t;
      INTPOL_INIT;
      INTPOL_3D(t, 1);

      /* Sedimentation velocity... */
      double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
                        atm->q[ctl->qnt_rhop][ip]);

```

```
      /* Calculate pressure change... */
      atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
    }
}
```

Here is the call graph for this function:



5.47.2.18 module_sort()

```
void module_sort (
    ctl_t * ctl,
    met_t * met0,
    atm_t * atm )
```

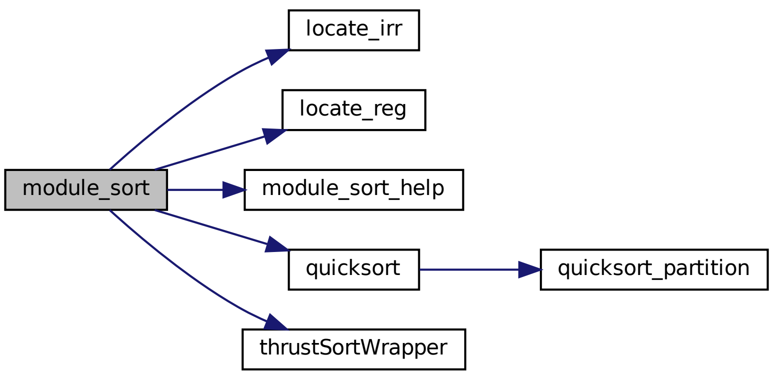
Sort particles according to box index.

Definition at line 1740 of file trac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("MODULE_SORT_BOXINDEX", "MEMORY", NVTX_GPU);

  /* Allocate... */
  const int np = atm->np;
  double *restrict const a = (double *) malloc((size_t) np * sizeof(double));
  int *restrict const p = (int *) malloc((size_t) np * sizeof(int));

#ifdef _OPENACC
#pragma acc enter data create(a[0:np],p[0:np])
#pragma acc data present(ctl,met0,atm,a,p)
#endif

  /* Get box index... */
#ifdef _OPENACC
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < np; ip++) {
    a[ip] =
      (double) ((locate_reg(met0->lon, met0->nx, atm->lon[ip]) * met0->ny +
                 locate_reg(met0->lat, met0->ny,
                            atm->lat[ip])) * met0->np + locate_irr(met0->p,
                                                                    met0->np,
                                                                    atm->p
                                                                    [ip]));
    p[ip] = ip;
  }

  /* Set timer... */
  SELECT_TIMER("MODULE_SORT_THRUST", "MEMORY", NVTX_GPU);

  /* Sorting... */
#ifdef _OPENACC
  {
#ifdef THRUST
    {
#pragma acc host_data use_device(a, p)
      thrustSortWrapper(a, np, p);
    }
#else
    {
```

```
#pragma acc update host(a[0:np], p[0:np])
#pragma omp parallel
      {
#pragma omp single nowait
        quicksort(a, p, 0, np - 1);
      }
#pragma acc update device(a[0:np], p[0:np])
    }
#endif
  }
#else
  {
#ifdef THRUST
    {
      thrustSortWrapper(a, np, p);
    }
#else
    {
#pragma omp parallel
      {
#pragma omp single nowait
        quicksort(a, p, 0, np - 1);
      }
    }
#endif
  }
#endif

  /* Set timer... */
  SELECT_TIMER("MODULE_SORT_REORDERING", "MEMORY", NVTX_GPU);

  /* Sort data... */
  module_sort_help(atm->time, p, np);
  module_sort_help(atm->p, p, np);
  module_sort_help(atm->lon, p, np);
  module_sort_help(atm->lat, p, np);
  for (int iq = 0; iq < ctl->nq; iq++)
    module_sort_help(atm->q[iq], p, np);

  /* Free... */
#ifdef _OPENACC
#pragma acc exit data delete(a,p)
#endif
  free(a);
  free(p);
}
```

Here is the call graph for this function:



5.47.2.19 module_sort_help() `void module_sort_help (`

`double * a,`

int * *p*,
int *np*)

Helper function for sorting module.

Definition at line 1837 of file trac.c.

```
          {

  /* Allocate... */
  double *restrict const help =
    (double *) malloc((size_t) np * sizeof(double));

  /* Reordering of array... */
#ifdef _OPENACC
#pragma acc enter data create(help[0:np])
#pragma acc data present(a,p,help)
#pragma acc parallel loop independent gang vector
#endif
  for (int ip = 0; ip < np; ip++)
    help[ip] = a[p[ip]];
#ifdef _OPENACC
#pragma acc parallel loop independent gang vector
#endif
  for (int ip = 0; ip < np; ip++)
    a[ip] = help[ip];

  /* Free... */
#ifdef _OPENACC
#pragma acc exit data delete(help)
#endif
  free(help);
}
```

5.47.2.20 module_timesteps()

void module_timesteps (
ctl_t * *ctl*,
atm_t * *atm*,
met_t * *met0*,
double * *dt*,
double *t*)

Calculate time steps.

Definition at line 1869 of file trac.c.

```
            {

  /* Set timer... */
  SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);

  const double latmin = gsl_stats_min(met0->lat, 1, (size_t) met0->ny),
    latmax = gsl_stats_max(met0->lat, 1, (size_t) met0->ny);

  const int np = atm->np,
    local = (fabs(met0->lon[met0->nx - 1] - met0->lon[0] - 360.0) >= 0.01);

#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 < np; ip++) {

    /* Set time step for each air parcel... */
    if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
         && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
         && ctl->direction * (atm->time[ip] - t) < 0))
      dt[ip] = t - atm->time[ip];
    else
      dt[ip] = 0.0;

    /* Check horizontal boundaries of local meteo data... */
    if (local && (atm->lon[ip] <= met0->lon[0]
                  || atm->lon[ip] >= met0->lon[met0->nx - 1]
                  || atm->lat[ip] <= latmin || atm->lat[ip] >= latmax))
      dt[ip] = 0.0;
  }
}
```

5.47.2.21 module_timesteps_init()

```
void module_timesteps_init (
    ctl_t * ctl,
    atm_t * atm )
```

Initialize timesteps.

Definition at line 1911 of file trac.c.

```
                  {

  /* Set timer... */
  SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);

  /* 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! Check T_STOP and DIRECTION!");

  /* 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;
}
```

5.47.2.22 module_wet_deposition()

```
void module_wet_deposition (
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )
```

Calculate wet deposition.

Definition at line 1942 of file trac.c.

```
                 {

  /* Set timer... */
  SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;

      /* Check whether particle is below cloud top... */
      INTPOL_INIT;
      INTPOL_2D(pct, 1);
      if (!isfinite(pct) || atm->p[ip] <= pct)
        continue;

      /* Get cloud bottom pressure... */
      INTPOL_2D(pcb, 0);

```

```
      /* Estimate precipitation rate (Pisso et al., 2019)... */
      INTPOL_2D(cl, 0);
      double Is =
        pow(1. / ctl->wet_depo_pre[0] * cl, 1. / ctl->wet_depo_pre[1]);
      if (Is < 0.01)
        continue;

      /* Check whether particle is inside or below cloud... */
      INTPOL_3D(lwc, 1);
      INTPOL_3D(iwc, 0);
      int inside = (iwc > 0 || lwc > 0);

      /* Get temperature... */
      INTPOL_3D(t, 0);

      /* Calculate in-cloud scavenging coefficient... */
      if (inside) {

        /* Calculate retention factor... */
        double eta;
        if (t > 273.15)
          eta = 1;
        else if (t <= 238.15)
          eta = ctl->wet_depo_ic_ret_ratio;
        else
          eta = LIN(273.15, 1, 238.15, ctl->wet_depo_ic_ret_ratio, t);

        /* Use exponential dependency for particles ... */
        if (ctl->wet_depo_ic_a > 0)
          lambda = ctl->wet_depo_ic_a * pow(Is, ctl->wet_depo_ic_b) * eta;

        /* Use Henry's law for gases... */
        else if (ctl->wet_depo_ic_h[0] > 0) {

          /* Get Henry's constant (Sander, 2015)... */
          h = ctl->wet_depo_ic_h[0]
            * exp(ctl->wet_depo_ic_h[1] * (1. / t - 1. / 298.15));

          /* Use effective Henry's constant for SO2
             (Berglen, 2004; Simpson, 2012)... */
          if (ctl->wet_depo_ic_h[2] > 0) {
            double H_ion = pow(10, ctl->wet_depo_ic_h[2] * (-1));
            double K_1 = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));
            double K_2 = 6e-8 * exp(1.12e3 * (1. / t - 1. / 298.15));
            h *= (1 + K_1 / H_ion + K_1 * K_2 / pow(H_ion, 2));
          }

          /* Estimate depth of cloud layer... */
          dz = 1e3 * (Z(pct) - Z(pcb));

          /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
          lambda = h * RI * t * Is / 3.6e6 / dz * eta;
        }
      }

      /* Calculate below-cloud scavenging coefficient... */
      else {

        /* Calculate retention factor... */
        double eta;
        if (t > 270)
          eta = 1;
        else
          eta = ctl->wet_depo_bc_ret_ratio;

        /* Use exponential dependency for particles... */
        if (ctl->wet_depo_bc_a > 0)
          lambda = ctl->wet_depo_bc_a * pow(Is, ctl->wet_depo_bc_b) * eta;

        /* Use Henry's law for gases... */
        else if (ctl->wet_depo_bc_h[0] > 0) {

          /* Get Henry's constant (Sander, 2015)... */
          h = ctl->wet_depo_bc_h[0]
            * exp(ctl->wet_depo_bc_h[1] * (1. / t - 1. / 298.15));

          /* Estimate depth of cloud layer... */
          dz = 1e3 * (Z(pct) - Z(pcb));

          /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
          lambda = h * RI * t * Is / 3.6e6 / dz * eta;
        }
      }

      /* Calculate exponential decay of mass... */
      double aux = exp(-dt[ip] * lambda);
      if (ctl->qnt_m >= 0) {
```

```
        if (ctl->qnt_mloss_wet >= 0)
          atm->q[ctl->qnt_mloss_wet][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}
```

5.47.2.23 write_output()

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

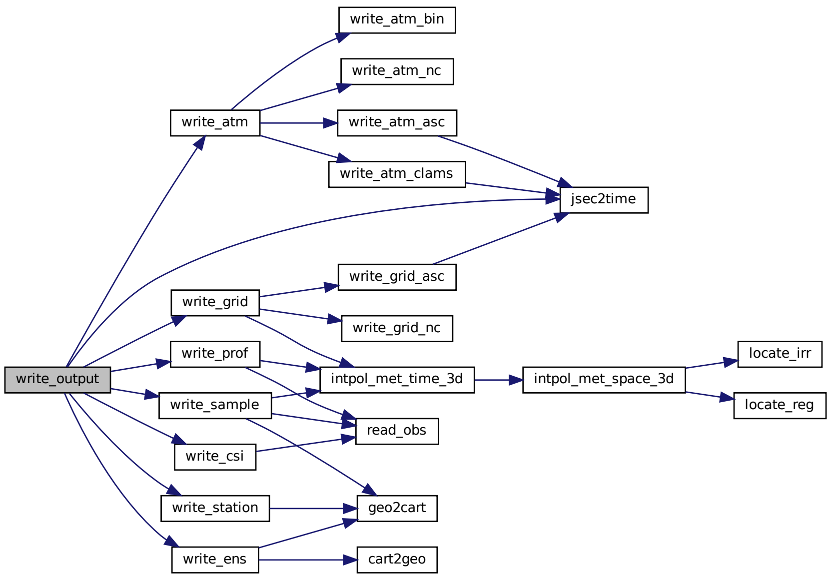
Write simulation output.

Definition at line 2076 of file trac.c.

```
                {

  char ext[10], filename[2 * LEN];

  double r;

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

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

  /* Update host... */
#ifdef _OPENACC
  if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
      || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
      || (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0)
      || ctl->csi_basename[0] != '-' || ctl->prof_basename[0] != '-'
      || ctl->sample_basename[0] != '-' || ctl->stat_basename[0] != '-') {
    SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
#pragma acc update host(atm[:1])
  }
#endif

  /* Write atmospheric data... */
  if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    if (ctl->atm_type == 0)
      sprintf(ext, "tab");
    else if (ctl->atm_type == 1)
      sprintf(ext, "bin");
    else if (ctl->atm_type == 2)
      sprintf(ext, "nc");
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
            dirname, ctl->atm_basename, year, mon, day, hour, min, ext);
    write_atm(filename, ctl, atm, t);
  }

  /* Write gridded data... */
  if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
            dirname, ctl->grid_basename, year, mon, day, hour, min,
            ctl->grid_type == 0 ? "tab" : "nc");
    write_grid(filename, ctl, met0, met1, atm, t);
  }

  /* Write CSI data... */
  if (ctl->csi_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
    write_csi(filename, ctl, atm, t);
  }

  /* Write ensemble data... */
  if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
            dirname, ctl->ens_basename, year, mon, day, hour, min);
    write_ens(filename, ctl, atm, t);
```

```
  }

  /* Write profile data... */
  if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write_prof(filename, ctl, met0, met1, atm, t);
  }

  /* Write sample data... */
  if (ctl->sample_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
    write_sample(filename, ctl, met0, met1, atm, t);
  }

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

Here is the call graph for this function:



5.47.2.24 main()

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

Definition at line 222 of file trac.c.

```
                  {

  ctl_t ctl;

  atm_t *atm;

  cache_t *cache;

  clim_t *clim;

  met_t *met0, *met1;

#ifdef ASYNCIO
  met_t *met0TMP, *met1TMP, *mets;
```

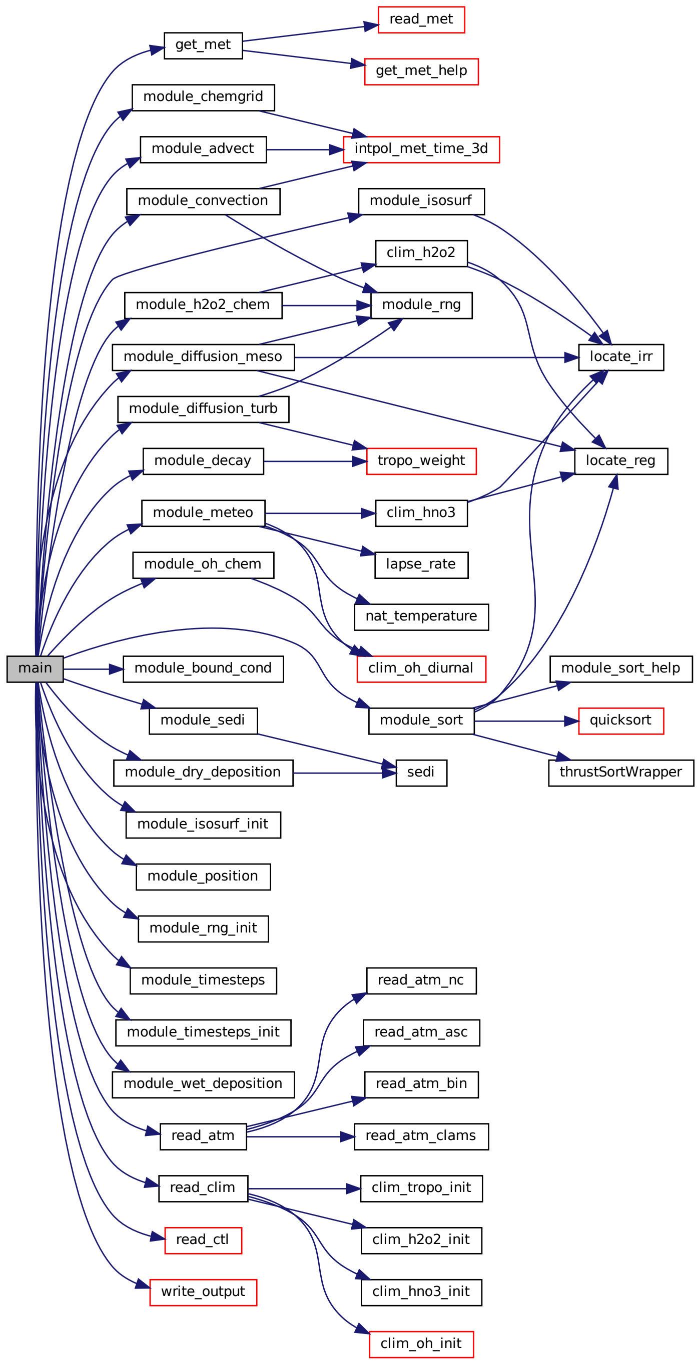
```
  ctl_t ctlTMP;
#endif

  FILE *dirlist;

  char dirname[LEN], filename[2 * LEN];

  double *dt, *rs, t;

  int ntask = -1, rank = 0, size = 1;

  /* Start timers... */
  START_TIMERS;

  /* Initialize MPI... */
#ifdef MPI
  SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
#endif

  /* Initialize GPUs... */
#ifdef _OPENACC
  SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
  if (acc_get_num_devices(acc_device_nvidia) <= 0)
    ERRMSG("Not running on a GPU device!");
  acc_set_device_num(rank % acc_get_num_devices(acc_device_nvidia),
                     acc_device_nvidia);
  acc_device_t device_type = acc_get_device_type();
  acc_init(device_type);
#endif

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

  /* Open directory list... */
  if (!(dirlist = fopen(argv[1], "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%4999s", dirname) != EOF) {

    /* MPI parallelization... */
    if ((++ntask) % size != rank)
      continue;

    /* ------------------------------------------------------------
       Initialize model run...
       ------------------------------------------------------------ */

    /* Allocate... */
    SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
    ALLOC(atm, atm_t, 1);
    ALLOC(cache, cache_t, 1);
    ALLOC(clim, clim_t, 1);
    ALLOC(met0, met_t, 1);
    ALLOC(met1, met_t, 1);
#ifdef ASYNCIO
    ALLOC(met0TMP, met_t, 1);
    ALLOC(met1TMP, met_t, 1);
#endif
    ALLOC(dt, double,
          NP);
    ALLOC(rs, double,
          3 * NP + 1);

    /* Create data region on GPUs... */
#ifdef _OPENACC
    SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
#ifdef ASYNCIO
#pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl,ctlTMP, met0[:1], met1[:1], met0TMP[:1], met1TMP[:1], dt[:NP], rs[:3 * NP])
#else
#pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3 * NP])
#endif
#endif

    /* Read control parameters... */
    sprintf(filename, "%s/%s", dirname, argv[2]);
    read_ctl(filename, argc, argv, &ctl);

    /* Read climatological data... */
    read_clim(&ctl, clim);

```

```
    /* Read atmospheric data... */
    sprintf(filename, "%s/%s", dirname, argv[3]);
    if (!read_atm(filename, &ctl, atm))
      ERRMSG("Cannot open file!");

    /* Initialize timesteps... */
    module_timesteps_init(&ctl, atm);

    /* Update GPU... */
#ifdef _OPENACC
    SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
#pragma acc update device(atm[:1], clim[:1], ctl)
#endif

    /* Initialize random number generator... */
    module_rng_init(ntask);

    /* Initialize meteo data... */
#ifdef ASYNCIO
    ctlTMP = ctl;
#endif
    get_met(&ctl, clim, 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!");
#ifdef ASYNCIO
    get_met(&ctlTMP, clim, ctlTMP.t_start, &met0TMP, &met1TMP);
#endif

    /* Initialize isosurface... */
    if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
      module_isosurf_init(&ctl, met0, met1, atm, cache);

    /* Update GPU... */
#ifdef _OPENACC
    SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
#pragma acc update device(cache[:1])
#endif

    /* ------------------------------------------------------------
       Loop over timesteps...
       ------------------------------------------------------------ */

    /* Loop over timesteps... */
#ifdef ASYNCIO
    omp_set_nested(1);
    // omp_set_dynamic(0);
    int ompTrdnum = omp_get_max_threads();
#endif
    for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
         t += ctl.direction * ctl.dt_mod) {
#ifdef ASYNCIO
#pragma omp parallel num_threads(2)
      {
#endif

        /* Adjust length of final time step... */
        if (ctl.direction * (t - ctl.t_stop) > 0)
          t = ctl.t_stop;

        /* Set time steps of air parcels... */
        module_timesteps(&ctl, atm, met0, dt, t);

        /* Get meteo data... */
#ifdef ASYNCIO
#pragma acc wait(5)
#pragma omp barrier
        if (omp_get_thread_num() == 0) {

          /* Pointer swap... */
          if (t != ctl.t_start) {
            mets = met0;
            met0 = met0TMP;
            met0TMP = mets;

            mets = met1;
            met1 = met1TMP;
            met1TMP = mets;
          }
#endif
#ifndef ASYNCIO
          if (t != ctl.t_start)
            get_met(&ctl, clim, t, &met0, &met1);
#endif

          /* Sort particles... */
          if (ctl.sort_dt > 0 && fmod(t, ctl.sort_dt) == 0)
            module_sort(&ctl, met0, atm);
```

```

          /* Check initial positions... */
          module_position(&ctl, met0, met1, atm, dt);

          /* Advection... */
          module_advect(&ctl, met0, met1, atm, dt);

          /* Turbulent diffusion... */
          if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
              || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
            module_diffusion_turb(&ctl, clim, atm, dt, rs);

          /* Mesoscale diffusion... */
          if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
            module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);

          /* Convection... */
          if (ctl.conv_cape >= 0
              && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
            module_convection(&ctl, met0, met1, atm, dt, rs);

          /* Sedimentation... */
          if (ctl.qnt_rp >= 0 && ctl.qnt_rhop >= 0)
            module_sedi(&ctl, met0, met1, atm, dt);

          /* Isosurface... */
          if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
            module_isosurf(&ctl, met0, met1, atm, cache);

          /* Check final positions... */
          module_position(&ctl, met0, met1, atm, dt);

          /* Interpolate meteo data... */
          if (ctl.met_dt_out > 0
              && (ctl.met_dt_out < ctl.dt_mod
                  || fmod(t, ctl.met_dt_out) == 0))
            module_meteo(&ctl, clim, met0, met1, atm);

          /* Decay of particle mass... */
          if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
            module_decay(&ctl, clim, atm, dt);

          /* OH chemistry... */
          if (ctl.clim_oh_filename[0] != '-' && ctl.oh_chem_reaction != 0)
            module_oh_chem(&ctl, clim, met0, met1, atm, dt);

          /* H2O2 chemistry (for SO2 aqueous phase oxidation)... */
          if (ctl.clim_h2o2_filename[0] != '-' && ctl.h2o2_chem_reaction != 0) {
            module_chemgrid(&ctl, met0, met1, atm, t);
            module_h2o2_chem(&ctl, clim, met0, met1, atm, dt, rs);
          }

          /* Dry deposition... */
          if (ctl.dry_depo_vdep > 0)
            module_dry_deposition(&ctl, met0, met1, atm, dt);

          /* Wet deposition... */
          if ((ctl.wet_depo_ic_a > 0 || ctl.wet_depo_ic_h[0] > 0)
              && (ctl.wet_depo_bc_a > 0 || ctl.wet_depo_bc_h[0] > 0))
            module_wet_deposition(&ctl, met0, met1, atm, dt);

          /* Boundary conditions... */
          if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
            module_bound_cond(&ctl, met0, met1, atm, dt);

          /* Write output... */
          write_output(dirname, &ctl, met0, met1, atm, t);
#ifdef ASYNCIO
        } else {
          omp_set_num_threads(ompTrdnum);
          if (ctl.direction * (t - ctl.t_stop + ctl.direction * ctl.dt_mod) <
              ctl.dt_mod)
            get_met(&ctl, clim, t + (ctl.direction * ctl.dt_mod), &met0TMP,
                    &met1TMP);
        }
      }
#endif
    }

#ifdef ASYNCIO
    omp_set_num_threads(ompTrdnum);
#endif

    /* ------------------------------------------------------------
       Finalize model run...
       ------------------------------------------------------------ */

```

```
    /* Report problem size... */
    LOG(1, "SIZE_NP = %d", atm->np);
    LOG(1, "SIZE_MPI_TASKS = %d", size);
    LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
#ifdef _OPENACC
    LOG(1, "SIZE_ACC_DEVICES = %d", acc_get_num_devices(acc_device_nvidia));
#endif

    /* Report memory usage... */
    LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
    LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
    LOG(1, "MEMORY_CLIM = %g MByte", sizeof(clim_t) / 1024. / 1024.);
    LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
    LOG(1, "MEMORY_DYNAMIC = %g MByte", (3 * NP * sizeof(int)
                                         + 4 * NP * sizeof(double)
                                         + EX * EY * EP * sizeof(float)) /
        1024. / 1024.);
    LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * EP * sizeof(float)) /
        1024. / 1024.);

    /* Delete data region on GPUs... */
#ifdef _OPENACC
    SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
#ifdef ASYNCIO
#pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs, met0TMP, met1TMP)
#else
#pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs)
#endif
#endif

    /* Free... */
    SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
    free(atm);
    free(cache);
    free(clim);
    free(met0);
    free(met1);
#ifdef ASYNCIO
    free(met0TMP);
    free(met1TMP);
#endif
    free(dt);
    free(rs);

    /* Report timers... */
    PRINT_TIMERS;
  }

  /* Finalize MPI... */
#ifdef MPI
  MPI_Finalize();
#endif

  /* Stop timers... */
  STOP_TIMERS;

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.48 trac.c

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

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

```

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

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

  Copyright (C) 2013-2023 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

/* ------------------------------------------------------------
   Global variables...
   ------------------------------------------------------------ */

#ifdef _OPENACC
curandGenerator_t rng;
#else
static gsl_rng *rng[NTHREADS];
#endif

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

void module_advect(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_bound_cond(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_convection(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt,
  double *rs);

void module_decay(
  ctl_t * ctl,
  clim_t * clim,
  atm_t * atm,
  double *dt);

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

void module_diffusion_turb(
  ctl_t * ctl,
  clim_t * clim,
  atm_t * atm,
  double *dt,
  double *rs);

void module_dry_deposition(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_isosurf_init(
  ctl_t * ctl,
  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,
  clim_t * clim,
  met_t * met0,
  met_t * met1,
  atm_t * atm);

void module_oh_chem(
  ctl_t * ctl,
  clim_t * clim,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_h2o2_chem(
  ctl_t * ctl,
  clim_t * clim,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt,
  double *rs);

void module_chemgrid(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double t);

void module_position(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_rng_init(
  int ntask);

void module_rng(
  double *rs,
  size_t n,
  int method);

void module_sedi(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt);

void module_sort(
  ctl_t * ctl,
  met_t * met0,
  atm_t * atm);

void module_sort_help(
  double *a,
  int *p,
  int np);

void module_timesteps(
  ctl_t * ctl,
  atm_t * atm,
  met_t * met0,
  double *dt,
  double t);

void module_timesteps_init(
  ctl_t * ctl,
  atm_t * atm);

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;

  clim_t *clim;

  met_t *met0, *met1;

#ifdef ASYNCIO
  met_t *met0TMP, *met1TMP, *mets;
  ctl_t ctlTMP;
#endif

  FILE *dirlist;

  char dirname[LEN], filename[2 * LEN];

  double *dt, *rs, t;

  int ntask = -1, rank = 0, size = 1;

  /* Start timers... */
  START_TIMERS;

  /* Initialize MPI... */
#ifdef MPI
  SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
#endif

  /* Initialize GPUs... */
#ifdef _OPENACC
  SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
  if (acc_get_num_devices(acc_device_nvidia) <= 0)
    ERRMSG("Not running on a GPU device!");
  acc_set_device_num(rank % acc_get_num_devices(acc_device_nvidia),
                     acc_device_nvidia);
  acc_device_t device_type = acc_get_device_type();
  acc_init(device_type);
#endif

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

  /* Open directory list... */
  if (!(dirlist = fopen(argv[1], "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%4999s", dirname) != EOF) {

    /* MPI parallelization... */
    if ((++ntask) % size != rank)
      continue;

    /* ------------------------------------------------------------
       Initialize model run...
       ------------------------------------------------------------ */

    /* Allocate... */
    SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
    ALLOC(atm, atm_t, 1);
    ALLOC(cache, cache_t, 1);
    ALLOC(clim, clim_t, 1);
    ALLOC(met0, met_t, 1);
    ALLOC(met1, met_t, 1);
#ifdef ASYNCIO
```

```
    ALLOC(met0TMP, met_t, 1);
    ALLOC(met1TMP, met_t, 1);
#endif
    ALLOC(dt, double,
          NP);
    ALLOC(rs, double,
          3 * NP + 1);

    /* Create data region on GPUs... */
#ifdef _OPENACC
    SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
#ifdef ASYNCIO
#pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl,ctlTMP, met0[:1], met1[:1], met0TMP[:1], met1TMP[:1], dt[:NP], rs[:3 * NP])
#else
#pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3 * NP])
#endif
#endif

    /* Read control parameters... */
    sprintf(filename, "%s/%s", dirname, argv[2]);
    read_ctl(filename, argc, argv, &ctl);

    /* Read climatological data... */
    read_clim(&ctl, clim);

    /* Read atmospheric data... */
    sprintf(filename, "%s/%s", dirname, argv[3]);
    if (!read_atm(filename, &ctl, atm))
      ERRMSG("Cannot open file!");

    /* Initialize timesteps... */
    module_timesteps_init(&ctl, atm);

    /* Update GPU... */
#ifdef _OPENACC
    SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
#pragma acc update device(atm[:1], clim[:1], ctl)
#endif

    /* Initialize random number generator... */
    module_rng_init(ntask);

    /* Initialize meteo data... */
#ifdef ASYNCIO
    ctlTMP = ctl;
#endif
    get_met(&ctl, clim, 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!");
#ifdef ASYNCIO
    get_met(&ctlTMP, clim, ctlTMP.t_start, &met0TMP, &met1TMP);
#endif

    /* Initialize isosurface... */
    if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
      module_isosurf_init(&ctl, met0, met1, atm, cache);

    /* Update GPU... */
#ifdef _OPENACC
    SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
#pragma acc update device(cache[:1])
#endif

    /* ------------------------------------------------------------
       Loop over timesteps...
       ------------------------------------------------------------ */

    /* Loop over timesteps... */
#ifdef ASYNCIO
    omp_set_nested(1);
    // omp_set_dynamic(0);
    int ompTrdnum = omp_get_max_threads();
#endif
    for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
         t += ctl.direction * ctl.dt_mod) {
#ifdef ASYNCIO
#pragma omp parallel num_threads(2)
      {
#endif

        /* Adjust length of final time step... */
        if (ctl.direction * (t - ctl.t_stop) > 0)
          t = ctl.t_stop;

        /* Set time steps of air parcels... */
```

```
      module_timesteps(&ctl, atm, met0, dt, t);

      /* Get meteo data... */
#ifdef ASYNCIO
#pragma acc wait(5)
#pragma omp barrier
      if (omp_get_thread_num() == 0) {

        /* Pointer swap... */
        if (t != ctl.t_start) {
          mets = met0;
          met0 = met0TMP;
          met0TMP = mets;

          mets = met1;
          met1 = met1TMP;
          met1TMP = mets;
        }
#endif
#ifndef ASYNCIO
        if (t != ctl.t_start)
          get_met(&ctl, clim, t, &met0, &met1);
#endif

        /* Sort particles... */
        if (ctl.sort_dt > 0 && fmod(t, ctl.sort_dt) == 0)
          module_sort(&ctl, met0, atm);

        /* Check initial positions... */
        module_position(&ctl, met0, met1, atm, dt);

        /* Advection... */
        module_advect(&ctl, met0, met1, atm, dt);

        /* Turbulent diffusion... */
        if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
            || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
          module_diffusion_turb(&ctl, clim, atm, dt, rs);

        /* Mesoscale diffusion... */
        if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
          module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);

        /* Convection... */
        if (ctl.conv_cape >= 0
            && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
          module_convection(&ctl, met0, met1, atm, dt, rs);

        /* Sedimentation... */
        if (ctl.qnt_rp >= 0 && ctl.qnt_rhop >= 0)
          module_sedi(&ctl, met0, met1, atm, dt);

        /* Isosurface... */
        if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
          module_isosurf(&ctl, met0, met1, atm, cache);

        /* Check final positions... */
        module_position(&ctl, met0, met1, atm, dt);

        /* Interpolate meteo data... */
        if (ctl.met_dt_out > 0
            && (ctl.met_dt_out < ctl.dt_mod
                || fmod(t, ctl.met_dt_out) == 0))
          module_meteo(&ctl, clim, met0, met1, atm);

        /* Decay of particle mass... */
        if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
          module_decay(&ctl, clim, atm, dt);

        /* OH chemistry... */
        if (ctl.clim_oh_filename[0] != '-' && ctl.oh_chem_reaction != 0)
          module_oh_chem(&ctl, clim, met0, met1, atm, dt);

        /* H2O2 chemistry (for SO2 aqueous phase oxidation)... */
        if (ctl.clim_h2o2_filename[0] != '-' && ctl.h2o2_chem_reaction != 0) {
          module_chemgrid(&ctl, met0, met1, atm, t);
          module_h2o2_chem(&ctl, clim, met0, met1, atm, dt, rs);
        }

        /* Dry deposition... */
        if (ctl.dry_depo_vdep > 0)
          module_dry_deposition(&ctl, met0, met1, atm, dt);

        /* Wet deposition... */
        if ((ctl.wet_depo_ic_a > 0 || ctl.wet_depo_ic_h[0] > 0)
            && (ctl.wet_depo_bc_a > 0 || ctl.wet_depo_bc_h[0] > 0))
          module_wet_deposition(&ctl, met0, met1, atm, dt);
```

```

          /* Boundary conditions... */
          if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
            module_bound_cond(&ctl, met0, met1, atm, dt);

          /* Write output... */
          write_output(dirname, &ctl, met0, met1, atm, t);
#ifdef ASYNCIO
        } else {
          omp_set_num_threads(ompTrdnum);
          if (ctl.direction * (t - ctl.t_stop + ctl.direction * ctl.dt_mod) <
              ctl.dt_mod)
            get_met(&ctl, clim, t + (ctl.direction * ctl.dt_mod), &met0TMP,
                    &met1TMP);
        }
      }
#endif
    }

#ifdef ASYNCIO
    omp_set_num_threads(ompTrdnum);
#endif

    /* ------------------------------------------------------------
       Finalize model run...
       ------------------------------------------------------------ */

    /* Report problem size... */
    LOG(1, "SIZE_NP = %d", atm->np);
    LOG(1, "SIZE_MPI_TASKS = %d", size);
    LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
#ifdef _OPENACC
    LOG(1, "SIZE_ACC_DEVICES = %d", acc_get_num_devices(acc_device_nvidia));
#endif

    /* Report memory usage... */
    LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
    LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
    LOG(1, "MEMORY_CLIM = %g MByte", sizeof(clim_t) / 1024. / 1024.);
    LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
    LOG(1, "MEMORY_DYNAMIC = %g MByte", (3 * NP * sizeof(int)
                                         + 4 * NP * sizeof(double)
                                         + EX * EY * EP * sizeof(float)) /
        1024. / 1024.);
    LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * EP * sizeof(float)) /
        1024. / 1024.);

    /* Delete data region on GPUs... */
#ifdef _OPENACC
    SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
#ifdef ASYNCIO
#pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs, met0TMP, met1TMP)
#else
#pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs)
#endif
#endif

    /* Free... */
    SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
    free(atm);
    free(cache);
    free(clim);
    free(met0);
    free(met1);
#ifdef ASYNCIO
    free(met0TMP);
    free(met1TMP);
#endif
    free(dt);
    free(rs);

    /* Report timers... */
    PRINT_TIMERS;
  }

  /* Finalize MPI... */
#ifdef MPI
  MPI_Finalize();
#endif

  /* Stop timers... */
  STOP_TIMERS;

  return EXIT_SUCCESS;
}

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

```
void module_advect(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_ADVECTION", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Init... */
      double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];

      /* Loop over integration nodes... */
      for (int i = 0; i < ctl->advect; i++) {

        /* Set position... */
        if (i == 0) {
          dts = 0.0;
          x[0] = atm->lon[ip];
          x[1] = atm->lat[ip];
          x[2] = atm->p[ip];
        } else {
          dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
          x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
          x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
          x[2] = atm->p[ip] + dts * w[i - 1];
        }
        double tm = atm->time[ip] + dts;

        /* Interpolate meteo data... */
#ifdef UVW
        intpol_met_time_uvw(met0, met1, tm, x[2], x[0], x[1],
                            &u[i], &v[i], &w[i]);
#else
        INTPOL_INIT;
        intpol_met_time_3d(met0, met0->u, met1, met1->u, tm,
                           x[2], x[0], x[1], &u[i], ci, cw, 1);
        intpol_met_time_3d(met0, met0->v, met1, met1->v, tm,
                           x[2], x[0], x[1], &v[i], ci, cw, 0);
        intpol_met_time_3d(met0, met0->w, met1, met1->w, tm,
                           x[2], x[0], x[1], &w[i], ci, cw, 0);
#endif

        /* Get mean wind... */
        double k = 1.0;
        if (ctl->advect == 2)
          k = (i == 0 ? 0.0 : 1.0);
        else if (ctl->advect == 4)
          k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
        um += k * u[i];
        vm += k * v[i];
        wm += k * w[i];
      }

      /* Set new position... */
      atm->time[ip] += dt[ip];
      atm->lon[ip] += DX2DEG(dt[ip] * um / 1000.,
                             (ctl->advect == 2 ? x[1] : atm->lat[ip]));
      atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
      atm->p[ip] += dt[ip] * wm;
    }
}

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

void module_bound_cond(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);

```

```
  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Check latitude and pressure range... */
      if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
          || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
        continue;

      /* Check surface layer... */
      if (ctl->bound_dps > 0 || ctl->bound_dzs > 0
          || ctl->bound_zetas > 0 || ctl->bound_pbl) {

        /* Get surface pressure... */
        double ps;
        INTPOL_INIT;
        INTPOL_2D(ps, 1);

        /* Check pressure... */
        if (ctl->bound_dps > 0 && atm->p[ip] < ps - ctl->bound_dps)
          continue;

        /* Check height... */
        if (ctl->bound_dzs > 0 && Z(atm->p[ip]) > Z(ps) + ctl->bound_dzs)
          continue;

        /* Check zeta range... */
        if (ctl->bound_zetas > 0) {
          double t;
          INTPOL_3D(t, 1);
          if (ZETA(ps, atm->p[ip], t) > ctl->bound_zetas)
            continue;
        }

        /* Check planetary boundary layer... */
        if (ctl->bound_pbl) {
          double pbl;
          INTPOL_2D(pbl, 0);
          if (atm->p[ip] < pbl)
            continue;
        }
      }

      /* Set mass and volume mixing ratio... */
      if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
        atm->q[ctl->qnt_m][ip] =
          ctl->bound_mass + ctl->bound_mass_trend * atm->time[ip];
      if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] =
          ctl->bound_vmr + ctl->bound_vmr_trend * atm->time[ip];
    }
}

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

void module_convection(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt,
  double *rs) {

  /* Set timer... */
  SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);

  /* Create random numbers... */
  module_rng(rs, (size_t) atm->np, 0);

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

```
    if (dt[ip] != 0) {

      double cape, cin, pel, ps;

      /* Interpolate CAPE... */
      INTPOL_INIT;
      INTPOL_2D(cape, 1);

      /* Check threshold... */
      if (isfinite(cape) && cape >= ctl->conv_cape) {

        /* Check CIN... */
        if (ctl->conv_cin > 0) {
          INTPOL_2D(cin, 0);
          if (isfinite(cin) && cin >= ctl->conv_cin)
            continue;
        }

        /* Interpolate equilibrium level... */
        INTPOL_2D(pel, 0);

        /* Check whether particle is above cloud top... */
        if (!isfinite(pel) || atm->p[ip] < pel)
          continue;

        /* Set pressure range for mixing... */
        double pbot = atm->p[ip];
        double ptop = atm->p[ip];
        if (ctl->conv_mix_bot == 1) {
          INTPOL_2D(ps, 0);
          pbot = ps;
        }
        if (ctl->conv_mix_top == 1)
          ptop = pel;

        /* Vertical mixing based on pressure... */
        if (ctl->conv_mix == 0)
          atm->p[ip] = pbot + (ptop - pbot) * rs[ip];

        /* Vertical mixing based on density... */
        else if (ctl->conv_mix == 1) {

          /* Get density range... */
          double tbot, ttop;
          intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                             pbot, atm->lon[ip], atm->lat[ip], &tbot,
                             ci, cw, 1);
          intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
                             ptop, atm->lon[ip], atm->lat[ip], &ttop,
                             ci, cw, 1);
          double rhobot = pbot / tbot;
          double rhotop = ptop / ttop;

          /* Get new density... */
          double lrho = log(rhobot + (rhotop - rhobot) * rs[ip]);

          /* Find pressure... */
          double lrhobot = log(rhobot);
          double lrhotop = log(rhotop);
          double lpbot = log(pbot);
          double lptop = log(ptop);
          atm->p[ip] = exp(LIN(lrhobot, lpbot, lrhotop, lptop, lrho));
        }
      }
    }
}

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

void module_decay(
  ctl_t * ctl,
  clim_t * clim,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#ifdef _OPENACC
#pragma acc data present(ctl,clim,atm,dt)
#pragma acc parallel loop independent gang vector
#else
```

```
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < np; ip++)
    if (dt[ip] != 0) {

      /* Get weighting factor... */
      double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);

      /* Set lifetime... */
      double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;

      /* Calculate exponential decay... */
      double aux = exp(-dt[ip] / tdec);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_decay >= 0)
          atm->q[ctl->qnt_mloss_decay][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}

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

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

  /* Set timer... */
  SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);

  /* Create random numbers... */
  module_rng(rs, 3 * (size_t) atm->np, 1);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* 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]);

      /* Get standard deviations of local wind data... */
      float umean = 0, usig = 0, vmean = 0, vsig = 0, wmean = 0, wsig = 0;
      for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
          for (int k = 0; k < 2; k++) {
#ifdef UVW
            umean += met0->uvw[ix + i][iy + j][iz + k][0];
            usig += SQR(met0->uvw[ix + i][iy + j][iz + k][0]);
            vmean += met0->uvw[ix + i][iy + j][iz + k][1];
            vsig += SQR(met0->uvw[ix + i][iy + j][iz + k][1]);
            wmean += met0->uvw[ix + i][iy + j][iz + k][2];
            wsig += SQR(met0->uvw[ix + i][iy + j][iz + k][2]);

            umean += met1->uvw[ix + i][iy + j][iz + k][0];
            usig += SQR(met1->uvw[ix + i][iy + j][iz + k][0]);
            vmean += met1->uvw[ix + i][iy + j][iz + k][1];
            vsig += SQR(met1->uvw[ix + i][iy + j][iz + k][1]);
            wmean += met1->uvw[ix + i][iy + j][iz + k][2];
            wsig += SQR(met1->uvw[ix + i][iy + j][iz + k][2]);
#else
            umean += met0->u[ix + i][iy + j][iz + k];
            usig += SQR(met0->u[ix + i][iy + j][iz + k]);
            vmean += met0->v[ix + i][iy + j][iz + k];
            vsig += SQR(met0->v[ix + i][iy + j][iz + k]);
            wmean += met0->w[ix + i][iy + j][iz + k];
            wsig += SQR(met0->w[ix + i][iy + j][iz + k]);

            umean += met1->u[ix + i][iy + j][iz + k];
            usig += SQR(met1->u[ix + i][iy + j][iz + k]);
            vmean += met1->v[ix + i][iy + j][iz + k];
            vsig += SQR(met1->v[ix + i][iy + j][iz + k]);
```

```
            wmean += met1->w[ix + i][iy + j][iz + k];
            wsig += SQR(met1->w[ix + i][iy + j][iz + k]);
#endif
          }
      usig = usig / 16.f - SQR(umean / 16.f);
      usig = (usig > 0 ? sqrtf(usig) : 0);
      vsig = vsig / 16.f - SQR(vmean / 16.f);
      vsig = (vsig > 0 ? sqrtf(vsig) : 0);
      wsig = wsig / 16.f - SQR(wmean / 16.f);
      wsig = (wsig > 0 ? sqrtf(wsig) : 0);

      /* 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->uvwp[ip][0] =
          (float) (r * cache->uvwp[ip][0] +
                   r2 * rs[3 * ip] * ctl->turb_mesox * usig);
        atm->lon[ip] +=
          DX2DEG(cache->uvwp[ip][0] * dt[ip] / 1000., atm->lat[ip]);

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

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

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

void module_diffusion_turb(
  ctl_t * ctl,
  clim_t * clim,
  atm_t * atm,
  double *dt,
  double *rs) {

  /* Set timer... */
  SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);

  /* Create random numbers... */
  module_rng(rs, 3 * (size_t) atm->np, 1);

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

      /* Get weighting factor... */
      double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], 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_dry_deposition(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      double ps, t, v_dep;

      /* Get surface pressure... */
      INTPOL_INIT;
      INTPOL_2D(ps, 1);

      /* Check whether particle is above the surface layer... */
      if (atm->p[ip] < ps - ctl->dry_depo_dp)
        continue;

      /* Set depth of surface layer... */
      double dz = 1000. * (Z(ps - ctl->dry_depo_dp) - Z(ps));

      /* Calculate sedimentation velocity for particles... */
      if (ctl->qnt_rp > 0 && ctl->qnt_rhop > 0) {

        /* Get temperature... */
        INTPOL_3D(t, 1);

        /* Set deposition velocity... */
        v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
                     atm->q[ctl->qnt_rhop][ip]);
      }

      /* Use explicit sedimentation velocity for gases... */
      else
        v_dep = ctl->dry_depo_vdep;

      /* Calculate loss of mass based on deposition velocity... */
      double aux = exp(-dt[ip] * v_dep / dz);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_dry >= 0)
          atm->q[ctl->qnt_mloss_dry][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}

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

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;

  /* Set timer... */
  SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);

  /* Init... */
  INTPOL_INIT;

```

```
  /* 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_3D(t, 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_3D(t, 1);
      cache->iso_var[ip] = THETA(atm->p[ip], t);
    }

  /* Read balloon pressure data... */
  else if (ctl->isosurf == 4) {

    /* Write info... */
    LOG(1, "Read balloon pressure data: %s", 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) {

  /* Set timer... */
  SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++) {

    double t;

    /* Init... */
    INTPOL_INIT;

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

    /* Restore density... */
    else if (ctl->isosurf == 2) {
      INTPOL_3D(t, 1);
      atm->p[ip] = cache->iso_var[ip] * t;
    }

    /* Restore potential temperature... */
    else if (ctl->isosurf == 3) {
      INTPOL_3D(t, 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,
  clim_t * clim,
  met_t * met0,
  met_t * met1,
  atm_t * atm) {

  /* Set timer... */
  SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);

  /* 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!");

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

    double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
      cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;

    /* Interpolate meteo data... */
    INTPOL_INIT;
    INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);

    /* Set quantities... */
    SET_ATM(qnt_ps, ps);
    SET_ATM(qnt_ts, ts);
    SET_ATM(qnt_zs, zs);
    SET_ATM(qnt_us, us);
    SET_ATM(qnt_vs, vs);
    SET_ATM(qnt_pbl, pbl);
    SET_ATM(qnt_pt, pt);
    SET_ATM(qnt_tt, tt);
    SET_ATM(qnt_zt, zt);
    SET_ATM(qnt_h2ot, h2ot);
    SET_ATM(qnt_z, z);
    SET_ATM(qnt_p, atm->p[ip]);
    SET_ATM(qnt_t, t);
    SET_ATM(qnt_rho, RHO(atm->p[ip], t));
    SET_ATM(qnt_u, u);
    SET_ATM(qnt_v, v);
    SET_ATM(qnt_w, w);
    SET_ATM(qnt_h2o, h2o);
    SET_ATM(qnt_o3, o3);
    SET_ATM(qnt_lwc, lwc);
    SET_ATM(qnt_iwc, iwc);
    SET_ATM(qnt_pct, pct);
    SET_ATM(qnt_pcb, pcb);
    SET_ATM(qnt_cl, cl);
    SET_ATM(qnt_plcl, plcl);
    SET_ATM(qnt_plfc, plfc);
    SET_ATM(qnt_pel, pel);
    SET_ATM(qnt_cape, cape);
    SET_ATM(qnt_cin, cin);
    SET_ATM(qnt_hno3,
            clim_hno3(clim, atm->time[ip], atm->lat[ip], atm->p[ip]));
    SET_ATM(qnt_oh,
            clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
                            atm->lon[ip], atm->lat[ip]));
    SET_ATM(qnt_vh, sqrt(u * u + v * v));
    SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
    SET_ATM(qnt_psat, PSAT(t));
```

```
    SET_ATM(qnt_psice, PSICE(t));
    SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
    SET_ATM(qnt_sh, SH(h2o));
    SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
    SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
    SET_ATM(qnt_theta, THETA(atm->p[ip], t));
    SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
    SET_ATM(qnt_tvirt, TVIRT(t, h2o));
    SET_ATM(qnt_lapse, lapse_rate(t, h2o));
    SET_ATM(qnt_pv, pv);
    SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
    SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
    SET_ATM(qnt_tnat,
            nat_temperature(atm->p[ip], h2o,
                            clim_hno3(clim, atm->time[ip], atm->lat[ip],
                                      atm->p[ip])));
    SET_ATM(qnt_tsts,
            0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
  }
}

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

void module_oh_chem(
  ctl_t * ctl,
  clim_t * clim,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

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

      /* Get temperature... */
      double t;
      INTPOL_INIT;
      INTPOL_3D(t, 1);

      /* Use constant reaction rate... */
      double k = GSL_NAN;
      if (ctl->oh_chem_reaction == 1)
        k = ctl->oh_chem[0];

      /* Calculate bimolecular reaction rate... */
      else if (ctl->oh_chem_reaction == 2)
        k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);

      /* Calculate termolecular reaction rate... */
      if (ctl->oh_chem_reaction == 3) {

        /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
        double M = 7.243e21 * (atm->p[ip] / 1000.) / t;

        /* Calculate rate coefficient for X + OH + M -> XOH + M
           (JPL Publication 19-05) ... */
        double k0 = ctl->oh_chem[0] *
          (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
        double ki = ctl->oh_chem[2] *
          (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
        double c = log10(k0 * M / ki);
        k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
      }

      /* Calculate exponential decay... */
      double rate_coef =
        k * clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
                            atm->lon[ip],
                            atm->lat[ip]);
      double aux = exp(-dt[ip] * rate_coef);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_oh >= 0)
          atm->q[ctl->qnt_mloss_oh][ip]
```

```
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}

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

void module_h2o2_chem(
  ctl_t * ctl,
  clim_t * clim,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt,
  double *rs) {

  /* Set timer... */
  SELECT_TIMER("MODULE_H2O2CHEM", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");
  if (ctl->qnt_vmrimpl < 0)
    ERRMSG("Module needs quantity implicit volume mixing ratio!");

  /* Create random numbers... */
  module_rng(rs, (size_t) atm->np, 0);

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

      /* Check whether particle is inside cloud... */
      double lwc;
      INTPOL_INIT;
      INTPOL_3D(lwc, 1);
      if (!(lwc > 0))
        continue;

      /* Check cloud cover... */
      if (rs[ip] > ctl->h2o2_chem_cc)
        continue;

      /* Check implicit volume mixing ratio... */
      if (atm->q[ctl->qnt_vmrimpl][ip] == 0)
        continue;

      /* Get temperature... */
      double t;
      INTPOL_3D(t, 0);

      /* Reaction rate (Berglen et al., 2004)... */
      double k = 9.1e7 * exp(-29700 / RI * (1. / t - 1. / 298.15));   // Maass  1999 unit: M^(-2)

      /* Henry constant of SO2... */
      double H_SO2 = 1.3e-2 * exp(2900 * (1. / t - 1. / 298.15)) * RI * t;
      double K_1S = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));     // unit: M

      /* Henry constant of H2O2... */
      double H_h2o2 = 8.3e2 * exp(7600 * (1 / t - 1 / 298.15)) * RI * t;

      /* Concentration of H2O2 (Barth et al., 1989)... */
      double SO2 = atm->q[ctl->qnt_vmrimpl][ip] * 1e9;      // vmr unit: ppbv
      double h2o2 = H_h2o2
        * clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip])
        * 0.59 * exp(-0.687 * SO2) * 1000 / 6.02214e23; // unit: M

      /* Volume water content in cloud [m^3 m^(-3)]... */
      double rho_air = 100 * atm->p[ip] / (RA * t);
      double CWC = lwc * rho_air / 1000;

      /* Calculate exponential decay (Rolph et al., 1992)... */
      double rate_coef = k * K_1S * h2o2 * H_SO2 * CWC;
      double aux = exp(-dt[ip] * rate_coef);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_h2o2 >= 0)
          atm->q[ctl->qnt_mloss_h2o2][ip] +=
            atm->q[ctl->qnt_m][ip] * (1 - aux);
```

```
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}

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

void module_chemgrid(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double t) {

  double *mass, *z, *lon, *lat, *press, *area;

  int *ixs, *iys, *izs;

  /* Update host... */
#ifdef _OPENACC
  SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
#pragma acc update host(atm[:1])
#endif

  /* Set timer... */
  SELECT_TIMER("MODULE_CHEMGRID", "PHYSICS", NVTX_GPU);

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

  /* Allocate... */
  ALLOC(mass, double,
        ctl->chemgrid_nx * ctl->chemgrid_ny * ctl->chemgrid_nz);
  ALLOC(z, double,
        ctl->chemgrid_nz);
  ALLOC(lon, double,
        ctl->chemgrid_nx);
  ALLOC(lat, double,
        ctl->chemgrid_ny);
  ALLOC(area, double,
        ctl->chemgrid_ny);
  ALLOC(press, double,
        ctl->chemgrid_nz);
  ALLOC(ixs, int,
        atm->np);
  ALLOC(iys, int,
        atm->np);
  ALLOC(izs, int,
        atm->np);

  /* Set grid box size... */
  double dz = (ctl->chemgrid_z1 - ctl->chemgrid_z0) / ctl->chemgrid_nz;
  double dlon = (ctl->chemgrid_lon1 - ctl->chemgrid_lon0) / ctl->chemgrid_nx;
  double dlat = (ctl->chemgrid_lat1 - ctl->chemgrid_lat0) / ctl->chemgrid_ny;

  /* Set vertical coordinates... */
#pragma omp parallel for default(shared)
  for (int iz = 0; iz < ctl->chemgrid_nz; iz++) {
    z[iz] = ctl->chemgrid_z0 + dz * (iz + 0.5);
    press[iz] = P(z[iz]);
  }

  /* Set horizontal coordinates... */
  for (int ix = 0; ix < ctl->chemgrid_nx; ix++)
    lon[ix] = ctl->chemgrid_lon0 + dlon * (ix + 0.5);
#pragma omp parallel for default(shared)
  for (int iy = 0; iy < ctl->chemgrid_ny; iy++) {
    lat[iy] = ctl->chemgrid_lat0 + dlat * (iy + 0.5);
    area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
      * cos(lat[iy] * M_PI / 180.);
  }

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

  /* Get indices... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++) {
    ixs[ip] = (int) ((atm->lon[ip] - ctl->chemgrid_lon0) / dlon);
    iys[ip] = (int) ((atm->lat[ip] - ctl->chemgrid_lat0) / dlat);
    izs[ip] = (int) ((Z(atm->p[ip]) - ctl->chemgrid_z0) / dz);
```

```
    if (atm->time[ip] < t0 || atm->time[ip] > t1
        || ixs[ip] < 0 || ixs[ip] >= ctl->chemgrid_nx
        || iys[ip] < 0 || iys[ip] >= ctl->chemgrid_ny
        || izs[ip] < 0 || izs[ip] >= ctl->chemgrid_nz)
      izs[ip] = -1;
  }

  /* Average data... */
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0)
      mass[ARRAY_3D
           (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
        += atm->q[ctl->qnt_m][ip];

  /* Interpolate volume mixing ratio... */
#pragma omp parallel for default(shared)
  for (int ip = 0; ip < atm->np; ip++)
    if (izs[ip] >= 0) {
      double temp;
      INTPOL_INIT;
      intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
                         lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
      atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass *
        mass[ARRAY_3D
             (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
        / (RHO(press[izs[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
    }

  /* Free... */
  free(mass);
  free(z);
  free(lon);
  free(lat);
  free(area);
  free(press);
  free(ixs);
  free(iys);
  free(izs);

  /* Update device... */
#ifdef _OPENACC
  SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
#pragma acc update device(atm[:1])
#endif
}

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

void module_position(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Init... */
      double ps;
      INTPOL_INIT;

      /* 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]) {
        if (ctl->reflect)
          atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
        else
          atm->p[ip] = met0->p[met0->np - 1];
      } else if (atm->p[ip] > 300.) {
        INTPOL_2D(ps, 1);
        if (atm->p[ip] > ps) {
          if (ctl->reflect)
            atm->p[ip] = 2. * ps - atm->p[ip];
          else
            atm->p[ip] = ps;
        }
      }
    }
}

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

void module_rng_init(
  int ntask) {

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

  if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT) !=
      CURAND_STATUS_SUCCESS)
    ERRMSG("Cannot create random number generator!");
  if (curandSetPseudoRandomGeneratorSeed(rng, ntask) != CURAND_STATUS_SUCCESS)
    ERRMSG("Cannot set seed for 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) (ntask * NTHREADS +
                                                        i));
  }

#endif
}

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

void module_rng(
  double *rs,
  size_t n,
  int method) {

#ifdef _OPENACC

#pragma acc host_data use_device(rs)
  {
    /* Uniform distribution... */
    if (method == 0) {
      if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n)) !=
          CURAND_STATUS_SUCCESS)
        ERRMSG("Cannot create random numbers!");
    }

    /* Normal distribution... */
    else if (method == 1) {
      if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0) !=
          CURAND_STATUS_SUCCESS)
        ERRMSG("Cannot create random numbers!");
    }
  }

#else

  /* Uniform distribution... */
  if (method == 0) {
```

```
#pragma omp parallel for default(shared)
    for (size_t i = 0; i < n; ++i)
      rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
  }

  /* Normal distribution... */
  else if (method == 1) {
#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_sedi(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

  /* Set timer... */
  SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      /* Get temperature... */
      double t;
      INTPOL_INIT;
      INTPOL_3D(t, 1);

      /* Sedimentation velocity... */
      double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
                        atm->q[ctl->qnt_rhop][ip]);

      /* Calculate pressure change... */
      atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
    }
}

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

void module_sort(
  ctl_t * ctl,
  met_t * met0,
  atm_t * atm) {

  /* Set timer... */
  SELECT_TIMER("MODULE_SORT_BOXINDEX", "MEMORY", NVTX_GPU);

  /* Allocate... */
  const int np = atm->np;
  double *restrict const a = (double *) malloc((size_t) np * sizeof(double));
  int *restrict const p = (int *) malloc((size_t) np * sizeof(int));

#ifdef _OPENACC
#pragma acc enter data create(a[0:np],p[0:np])
#pragma acc data present(ctl,met0,atm,a,p)
#endif

  /* Get box index... */
#ifdef _OPENACC
#pragma acc parallel loop independent gang vector
#else
#pragma omp parallel for default(shared)
#endif
  for (int ip = 0; ip < np; ip++) {
    a[ip] =
      (double) ((locate_reg(met0->lon, met0->nx, atm->lon[ip]) * met0->ny +
                 locate_reg(met0->lat, met0->ny,
                            atm->lat[ip])) * met0->np + locate_irr(met0->p,
                                                                   met0->np,
                                                                   atm->p
                                                                   [ip]));
    p[ip] = ip;
  }

```

```
  /* Set timer... */
  SELECT_TIMER("MODULE_SORT_THRUST", "MEMORY", NVTX_GPU);

  /* Sorting... */
#ifdef _OPENACC
  {
#ifdef THRUST
    {
#pragma acc host_data use_device(a, p)
      thrustSortWrapper(a, np, p);
    }
#else
    {
#pragma acc update host(a[0:np], p[0:np])
#pragma omp parallel
      {
#pragma omp single nowait
        quicksort(a, p, 0, np - 1);
      }
#pragma acc update device(a[0:np], p[0:np])
    }
#endif
  }
#else
  {
#ifdef THRUST
    {
      thrustSortWrapper(a, np, p);
    }
#else
    {
#pragma omp parallel
      {
#pragma omp single nowait
        quicksort(a, p, 0, np - 1);
      }
    }
#endif
  }
#endif

  /* Set timer... */
  SELECT_TIMER("MODULE_SORT_REORDERING", "MEMORY", NVTX_GPU);

  /* Sort data... */
  module_sort_help(atm->time, p, np);
  module_sort_help(atm->p, p, np);
  module_sort_help(atm->lon, p, np);
  module_sort_help(atm->lat, p, np);
  for (int iq = 0; iq < ctl->nq; iq++)
    module_sort_help(atm->q[iq], p, np);

  /* Free... */
#ifdef _OPENACC
#pragma acc exit data delete(a,p)
#endif
  free(a);
  free(p);
}

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

void module_sort_help(
  double *a,
  int *p,
  int np) {

  /* Allocate... */
  double *restrict const help =
    (double *) malloc((size_t) np * sizeof(double));

  /* Reordering of array... */
#ifdef _OPENACC
#pragma acc enter data create(help[0:np])
#pragma acc data present(a,p,help)
#pragma acc parallel loop independent gang vector
#endif
  for (int ip = 0; ip < np; ip++)
    help[ip] = a[p[ip]];
#ifdef _OPENACC
#pragma acc parallel loop independent gang vector
#endif
  for (int ip = 0; ip < np; ip++)
    a[ip] = help[ip];

  /* Free... */
#ifdef _OPENACC
```

```
#pragma acc exit data delete(help)
#endif
  free(help);
}

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

void module_timesteps(
  ctl_t * ctl,
  atm_t * atm,
  met_t * met0,
  double *dt,
  double t) {

  /* Set timer... */
  SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);

  const double latmin = gsl_stats_min(met0->lat, 1, (size_t) met0->ny),
    latmax = gsl_stats_max(met0->lat, 1, (size_t) met0->ny);

  const int np = atm->np,
    local = (fabs(met0->lon[met0->nx - 1] - met0->lon[0] - 360.0) >= 0.01);

#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 < np; ip++) {

    /* Set time step for each air parcel... */
    if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
         && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
         && ctl->direction * (atm->time[ip] - t) < 0))
      dt[ip] = t - atm->time[ip];
    else
      dt[ip] = 0.0;

    /* Check horizontal boundaries of local meteo data... */
    if (local && (atm->lon[ip] <= met0->lon[0]
                  || atm->lon[ip] >= met0->lon[met0->nx - 1]
                  || atm->lat[ip] <= latmin || atm->lat[ip] >= latmax))
      dt[ip] = 0.0;
  }
}

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

void module_timesteps_init(
  ctl_t * ctl,
  atm_t * atm) {

  /* Set timer... */
  SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);

  /* 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! Check T_STOP and DIRECTION!");

  /* 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;
}

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

void module_wet_deposition(
  ctl_t * ctl,
  met_t * met0,
  met_t * met1,
  atm_t * atm,
  double *dt) {

```

```
  /* Set timer... */
  SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);

  /* Check quantity flags... */
  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
    ERRMSG("Module needs quantity mass or volume mixing ratio!");

  const int np = atm->np;
#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 < np; ip++)
    if (dt[ip] != 0) {

      double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;

      /* Check whether particle is below cloud top... */
      INTPOL_INIT;
      INTPOL_2D(pct, 1);
      if (!isfinite(pct) || atm->p[ip] <= pct)
        continue;

      /* Get cloud bottom pressure... */
      INTPOL_2D(pcb, 0);

      /* Estimate precipitation rate (Pisso et al., 2019)... */
      INTPOL_2D(cl, 0);
      double Is =
        pow(1. / ctl->wet_depo_pre[0] * cl, 1. / ctl->wet_depo_pre[1]);
      if (Is < 0.01)
        continue;

      /* Check whether particle is inside or below cloud... */
      INTPOL_3D(lwc, 1);
      INTPOL_3D(iwc, 0);
      int inside = (iwc > 0 || lwc > 0);

      /* Get temperature... */
      INTPOL_3D(t, 0);

      /* Calculate in-cloud scavenging coefficient... */
      if (inside) {

        /* Calculate retention factor... */
        double eta;
        if (t > 273.15)
          eta = 1;
        else if (t <= 238.15)
          eta = ctl->wet_depo_ic_ret_ratio;
        else
          eta = LIN(273.15, 1, 238.15, ctl->wet_depo_ic_ret_ratio, t);

        /* Use exponential dependency for particles ... */
        if (ctl->wet_depo_ic_a > 0)
          lambda = ctl->wet_depo_ic_a * pow(Is, ctl->wet_depo_ic_b) * eta;

        /* Use Henry's law for gases... */
        else if (ctl->wet_depo_ic_h[0] > 0) {

          /* Get Henry's constant (Sander, 2015)... */
          h = ctl->wet_depo_ic_h[0]
            * exp(ctl->wet_depo_ic_h[1] * (1. / t - 1. / 298.15));

          /* Use effective Henry's constant for SO2
             (Berglen, 2004; Simpson, 2012)... */
          if (ctl->wet_depo_ic_h[2] > 0) {
            double H_ion = pow(10, ctl->wet_depo_ic_h[2] * (-1));
            double K_1 = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));
            double K_2 = 6e-8 * exp(1.12e3 * (1. / t - 1. / 298.15));
            h *= (1 + K_1 / H_ion + K_1 * K_2 / pow(H_ion, 2));
          }

          /* Estimate depth of cloud layer... */
          dz = 1e3 * (Z(pct) - Z(pcb));

          /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
          lambda = h * RI * t * Is / 3.6e6 / dz * eta;
        }
      }

      /* Calculate below-cloud scavenging coefficient... */
      else {

        /* Calculate retention factor... */
```

```
        double eta;
        if (t > 270)
          eta = 1;
        else
          eta = ctl->wet_depo_bc_ret_ratio;

        /* Use exponential dependency for particles... */
        if (ctl->wet_depo_bc_a > 0)
          lambda = ctl->wet_depo_bc_a * pow(Is, ctl->wet_depo_bc_b) * eta;

        /* Use Henry's law for gases... */
        else if (ctl->wet_depo_bc_h[0] > 0) {

          /* Get Henry's constant (Sander, 2015)... */
          h = ctl->wet_depo_bc_h[0]
            * exp(ctl->wet_depo_bc_h[1] * (1. / t - 1. / 298.15));

          /* Estimate depth of cloud layer... */
          dz = 1e3 * (Z(pct) - Z(pcb));

          /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
          lambda = h * RI * t * Is / 3.6e6 / dz * eta;
        }
      }

      /* Calculate exponential decay of mass... */
      double aux = exp(-dt[ip] * lambda);
      if (ctl->qnt_m >= 0) {
        if (ctl->qnt_mloss_wet >= 0)
          atm->q[ctl->qnt_mloss_wet][ip]
            += atm->q[ctl->qnt_m][ip] * (1 - aux);
        atm->q[ctl->qnt_m][ip] *= aux;
      }
      if (ctl->qnt_vmr >= 0)
        atm->q[ctl->qnt_vmr][ip] *= aux;
    }
}

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

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

  char ext[10], filename[2 * LEN];

  double r;

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

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

  /* Update host... */
#ifdef _OPENACC
  if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
      || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
      || (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0)
      || ctl->csi_basename[0] != '-' || ctl->prof_basename[0] != '-'
      || ctl->sample_basename[0] != '-' || ctl->stat_basename[0] != '-') {
    SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
#pragma acc update host(atm[:1])
  }
#endif

  /* Write atmospheric data... */
  if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
    if (ctl->atm_type == 0)
      sprintf(ext, "tab");
    else if (ctl->atm_type == 1)
      sprintf(ext, "bin");
    else if (ctl->atm_type == 2)
      sprintf(ext, "nc");
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
            dirname, ctl->atm_basename, year, mon, day, hour, min, ext);
    write_atm(filename, ctl, atm, t);
  }

  /* Write gridded data... */
  if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
            dirname, ctl->grid_basename, year, mon, day, hour, min,
            ctl->grid_type == 0 ? "tab" : "nc");
```

```
    write_grid(filename, ctl, met0, met1, atm, t);
  }

  /* Write CSI data... */
  if (ctl->csi_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
    write_csi(filename, ctl, atm, t);
  }

  /* Write ensemble data... */
  if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
    sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
            dirname, ctl->ens_basename, year, mon, day, hour, min);
    write_ens(filename, ctl, atm, t);
  }

  /* Write profile data... */
  if (ctl->prof_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
    write_prof(filename, ctl, met0, met1, atm, t);
  }

  /* Write sample data... */
  if (ctl->sample_basename[0] != '-') {
    sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
    write_sample(filename, ctl, met0, met1, atm, t);
  }

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

5.49 tropo.c File Reference

Create tropopause data set from meteorological data.

```
#include "libtrac.h"
```

Functions

- void get_tropo (int met_tropo, ctl_t *ctl, clim_t *clim, met_t *met, double *lons, int nx, double *lats, int ny, double *pt, double *zt, double *tt, double *qt, double *o3t, double *ps, double *zs)
- int main (int argc, char *argv[ ])

5.49.1 Detailed Description

Create tropopause data set from meteorological data.

Definition in file tropo.c.

5.49.2 Function Documentation

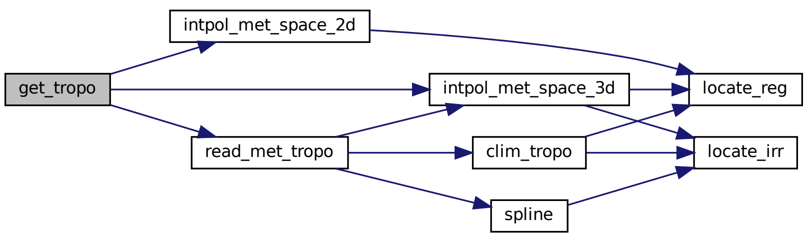
5.49.2.1 get_tropo()

```
void get_tropo (
        int met_tropo,
        ctl_t * ctl,
        clim_t * clim,
        met_t * met,
        double * lons,
        int nx,
        double * lats,
        int ny,
        double * pt,
        double * zt,
        double * tt,
        double * qt,
        double * o3t,
        double * ps,
        double * zs )
```

Definition at line 279 of file tropo.c.

```
                  {

  INTPOL_INIT;

  ctl->met_tropo = met_tropo;
  read_met_tropo(ctl, clim, met);
#pragma omp parallel for default(shared) private(ci,cw)
  for (int ix = 0; ix < nx; ix++)
    for (int 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_2d(met, met->ps, lons[ix], lats[iy],
                          &ps[iy * nx + ix], ci, cw, 0);
      intpol_met_space_2d(met, met->zs, lons[ix], lats[iy],
                          &zs[iy * nx + ix], ci, cw, 0);
      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);
      intpol_met_space_3d(met, met->o3, pt[iy * nx + ix], lons[ix],
                          lats[iy], &o3t[iy * nx + ix], ci, cw, 0);
    }
}
```

Here is the call graph for this function:



5.49.2.2 main()

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

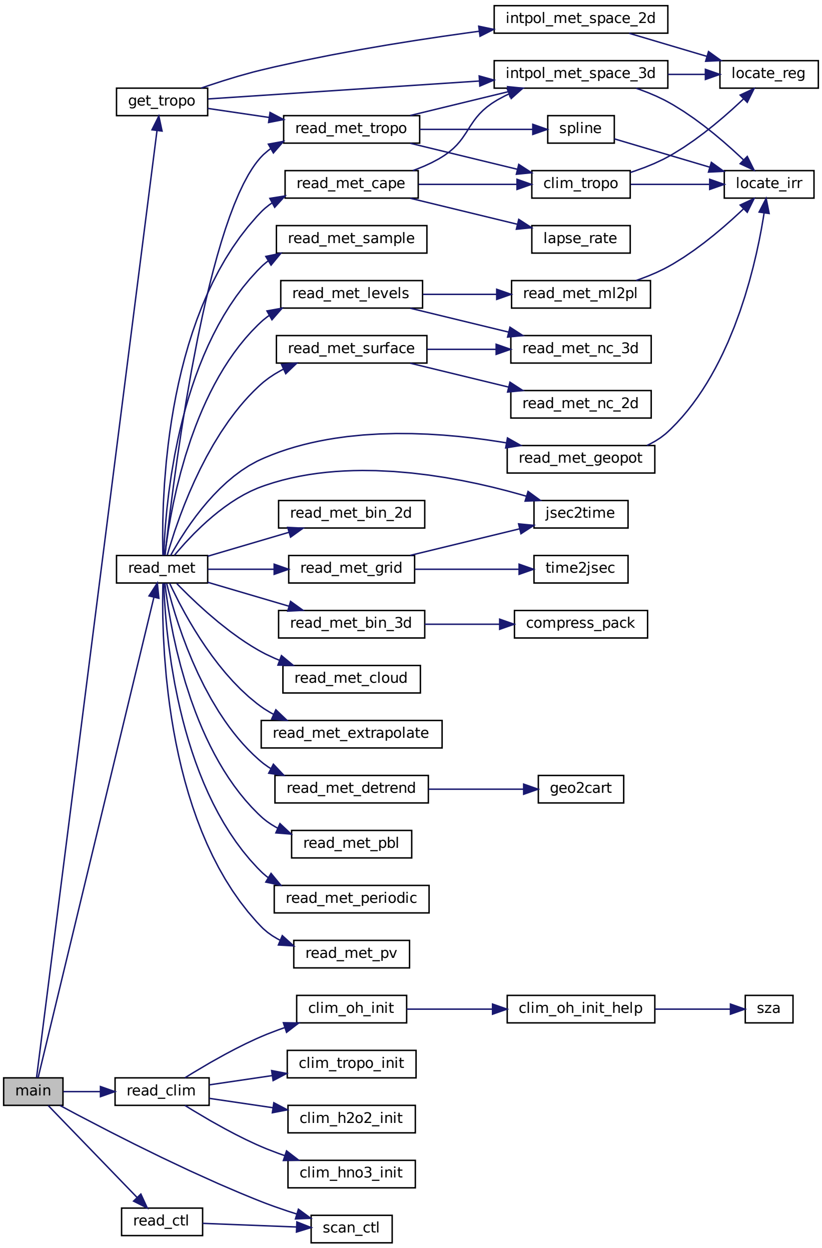
Definition at line 52 of file tropo.c.

```
                    {

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  static double ps[EX * EY], pt[EX * EY], qt[EX * EY], o3t[EX * EY],
    zs[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0, lon1, lons[EX], dlon,
    lat, lat0, lat1, lats[EY], dlat;

  static int init, i, nx, ny, nt, ncid, varid, dims[3], h2o, o3;

  static size_t count[10], start[10];

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  o3 = (int) scan_ctl(argv[1], argc, argv, "TROPO_O3", -1, "1", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    ctl.met_tropo = 0;
    if (!read_met(argv[i], &ctl, clim, 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... */
      LOG(1, "Write tropopause data file: %s", 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_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time",
```

```
                 "seconds since 2000-01-01 00:00:00 UTC");
      NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[1], "latitude", "degrees_north");
      NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[2], "longitude", "degrees_east");

      NC_DEF_VAR("clp_z", NC_FLOAT, 3, &dims[0], "cold point height", "km");
      NC_DEF_VAR("clp_p", NC_FLOAT, 3, &dims[0], "cold point pressure",
                 "hPa");
      NC_DEF_VAR("clp_t", NC_FLOAT, 3, &dims[0], "cold point temperature",
                 "K");
      if (h2o)
        NC_DEF_VAR("clp_q", NC_FLOAT, 3, &dims[0], "cold point water vapor",
                   "ppv");
      if (o3)
        NC_DEF_VAR("clp_o3", NC_FLOAT, 3, &dims[0], "cold point ozone",
                   "ppv");

      NC_DEF_VAR("dyn_z", NC_FLOAT, 3, &dims[0],
                 "dynamical tropopause height", "km");
      NC_DEF_VAR("dyn_p", NC_FLOAT, 3, &dims[0],
                 "dynamical tropopause pressure", "hPa");
      NC_DEF_VAR("dyn_t", NC_FLOAT, 3, &dims[0],
                 "dynamical tropopause temperature", "K");
      if (h2o)
        NC_DEF_VAR("dyn_q", NC_FLOAT, 3, &dims[0],
                   "dynamical tropopause water vapor", "ppv");
      if (o3)
        NC_DEF_VAR("dyn_o3", NC_FLOAT, 3, &dims[0],
                   "dynamical tropopause ozone", "ppv");

      NC_DEF_VAR("wmo_1st_z", NC_FLOAT, 3, &dims[0],
                 "WMO 1st tropopause height", "km");
      NC_DEF_VAR("wmo_1st_p", NC_FLOAT, 3, &dims[0],
                 "WMO 1st tropopause pressure", "hPa");
      NC_DEF_VAR("wmo_1st_t", NC_FLOAT, 3, &dims[0],
                 "WMO 1st tropopause temperature", "K");
      if (h2o)
        NC_DEF_VAR("wmo_1st_q", NC_FLOAT, 3, &dims[0],
                   "WMO 1st tropopause water vapor", "ppv");
      if (o3)
        NC_DEF_VAR("wmo_1st_o3", NC_FLOAT, 3, &dims[0],
                   "WMO 1st tropopause ozone", "ppv");

      NC_DEF_VAR("wmo_2nd_z", NC_FLOAT, 3, &dims[0],
                 "WMO 2nd tropopause height", "km");
      NC_DEF_VAR("wmo_2nd_p", NC_FLOAT, 3, &dims[0],
                 "WMO 2nd tropopause pressure", "hPa");
      NC_DEF_VAR("wmo_2nd_t", NC_FLOAT, 3, &dims[0],
                 "WMO 2nd tropopause temperature", "K");
      if (h2o)
        NC_DEF_VAR("wmo_2nd_q", NC_FLOAT, 3, &dims[0],
                   "WMO 2nd tropopause water vapor", "ppv");
      if (o3)
        NC_DEF_VAR("wmo_2nd_o3", NC_FLOAT, 3, &dims[0],
                   "WMO 2nd tropopause ozone", "ppv");

      NC_DEF_VAR("ps", NC_FLOAT, 3, &dims[0], "surface pressure", "hPa");
      NC_DEF_VAR("zs", NC_FLOAT, 3, &dims[0], "surface height", "km");

      /* End definition... */
      NC(nc_enddef(ncid));

      /* Write longitude and latitude... */
      NC_PUT_DOUBLE("lat", lats, 0);
      NC_PUT_DOUBLE("lon", lons, 0);
    }

    /* 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_PUT_DOUBLE("time", &met->time, 1);

    /* Get cold point... */
    get_tropo(2, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("clp_z", zt, 1);
    NC_PUT_DOUBLE("clp_p", pt, 1);
    NC_PUT_DOUBLE("clp_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("clp_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("clp_o3", o3t, 1);

    /* Get dynamical tropopause... */
```

```
    get_tropo(5, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("dyn_z", zt, 1);
    NC_PUT_DOUBLE("dyn_p", pt, 1);
    NC_PUT_DOUBLE("dyn_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("dyn_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("dyn_o3", o3t, 1);

    /* Get WMO 1st tropopause... */
    get_tropo(3, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("wmo_1st_z", zt, 1);
    NC_PUT_DOUBLE("wmo_1st_p", pt, 1);
    NC_PUT_DOUBLE("wmo_1st_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("wmo_1st_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("wmo_1st_o3", o3t, 1);

    /* Get WMO 2nd tropopause... */
    get_tropo(4, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("wmo_2nd_z", zt, 1);
    NC_PUT_DOUBLE("wmo_2nd_p", pt, 1);
    NC_PUT_DOUBLE("wmo_2nd_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("wmo_2nd_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("wmo_2nd_o3", o3t, 1);

    /* Write surface data... */
    NC_PUT_DOUBLE("ps", ps, 1);
    NC_PUT_DOUBLE("zs", zs, 1);

    /* Increment time step counter... */
    nt++;
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.50 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) 2013-2023 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

void get_tropo(
  int met_tropo,
  ctl_t * ctl,
  clim_t * clim,
  met_t * met,
  double *lons,
  int nx,
  double *lats,
  int ny,
  double *pt,
  double *zt,
  double *tt,
  double *qt,
  double *o3t,
  double *ps,
  double *zs);

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

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

  ctl_t ctl;

  clim_t *clim;

  met_t *met;

  static double ps[EX * EY], pt[EX * EY], qt[EX * EY], o3t[EX * EY],
    zs[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0, lon1, lons[EX], dlon,
    lat, lat0, lat1, lats[EY], dlat;

  static int init, i, nx, ny, nt, ncid, varid, dims[3], h2o, o3;

  static size_t count[10], start[10];

  /* Allocate... */
  ALLOC(clim, clim_t, 1);
  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);
  o3 = (int) scan_ctl(argv[1], argc, argv, "TROPO_O3", -1, "1", NULL);

  /* Read climatological data... */
  read_clim(&ctl, clim);

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

    /* Read meteorological data... */
    ctl.met_tropo = 0;
    if (!read_met(argv[i], &ctl, clim, 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... */
    LOG(1, "Write tropopause data file: %s", 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_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time",
               "seconds since 2000-01-01 00:00:00 UTC");
    NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[1], "latitude", "degrees_north");
    NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[2], "longitude", "degrees_east");

    NC_DEF_VAR("clp_z", NC_FLOAT, 3, &dims[0], "cold point height", "km");
    NC_DEF_VAR("clp_p", NC_FLOAT, 3, &dims[0], "cold point pressure",
               "hPa");
    NC_DEF_VAR("clp_t", NC_FLOAT, 3, &dims[0], "cold point temperature",
               "K");
    if (h2o)
      NC_DEF_VAR("clp_q", NC_FLOAT, 3, &dims[0], "cold point water vapor",
                 "ppv");
    if (o3)
      NC_DEF_VAR("clp_o3", NC_FLOAT, 3, &dims[0], "cold point ozone",
                 "ppv");

    NC_DEF_VAR("dyn_z", NC_FLOAT, 3, &dims[0],
               "dynamical tropopause height", "km");
    NC_DEF_VAR("dyn_p", NC_FLOAT, 3, &dims[0],
               "dynamical tropopause pressure", "hPa");
    NC_DEF_VAR("dyn_t", NC_FLOAT, 3, &dims[0],
               "dynamical tropopause temperature", "K");
    if (h2o)
      NC_DEF_VAR("dyn_q", NC_FLOAT, 3, &dims[0],
                 "dynamical tropopause water vapor", "ppv");
    if (o3)
      NC_DEF_VAR("dyn_o3", NC_FLOAT, 3, &dims[0],
                 "dynamical tropopause ozone", "ppv");

    NC_DEF_VAR("wmo_1st_z", NC_FLOAT, 3, &dims[0],
               "WMO 1st tropopause height", "km");
    NC_DEF_VAR("wmo_1st_p", NC_FLOAT, 3, &dims[0],
               "WMO 1st tropopause pressure", "hPa");
    NC_DEF_VAR("wmo_1st_t", NC_FLOAT, 3, &dims[0],
               "WMO 1st tropopause temperature", "K");
    if (h2o)
      NC_DEF_VAR("wmo_1st_q", NC_FLOAT, 3, &dims[0],
                 "WMO 1st tropopause water vapor", "ppv");
    if (o3)
      NC_DEF_VAR("wmo_1st_o3", NC_FLOAT, 3, &dims[0],
                 "WMO 1st tropopause ozone", "ppv");

    NC_DEF_VAR("wmo_2nd_z", NC_FLOAT, 3, &dims[0],
               "WMO 2nd tropopause height", "km");
    NC_DEF_VAR("wmo_2nd_p", NC_FLOAT, 3, &dims[0],
               "WMO 2nd tropopause pressure", "hPa");
    NC_DEF_VAR("wmo_2nd_t", NC_FLOAT, 3, &dims[0],
               "WMO 2nd tropopause temperature", "K");
    if (h2o)
      NC_DEF_VAR("wmo_2nd_q", NC_FLOAT, 3, &dims[0],
                 "WMO 2nd tropopause water vapor", "ppv");
    if (o3)
      NC_DEF_VAR("wmo_2nd_o3", NC_FLOAT, 3, &dims[0],
                 "WMO 2nd tropopause ozone", "ppv");
```

```
      NC_DEF_VAR("ps", NC_FLOAT, 3, &dims[0], "surface pressure", "hPa");
      NC_DEF_VAR("zs", NC_FLOAT, 3, &dims[0], "surface height", "km");

      /* End definition... */
      NC(nc_enddef(ncid));

      /* Write longitude and latitude... */
      NC_PUT_DOUBLE("lat", lats, 0);
      NC_PUT_DOUBLE("lon", lons, 0);
    }

    /* 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_PUT_DOUBLE("time", &met->time, 1);

    /* Get cold point... */
    get_tropo(2, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("clp_z", zt, 1);
    NC_PUT_DOUBLE("clp_p", pt, 1);
    NC_PUT_DOUBLE("clp_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("clp_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("clp_o3", o3t, 1);

    /* Get dynamical tropopause... */
    get_tropo(5, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("dyn_z", zt, 1);
    NC_PUT_DOUBLE("dyn_p", pt, 1);
    NC_PUT_DOUBLE("dyn_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("dyn_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("dyn_o3", o3t, 1);

    /* Get WMO 1st tropopause... */
    get_tropo(3, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("wmo_1st_z", zt, 1);
    NC_PUT_DOUBLE("wmo_1st_p", pt, 1);
    NC_PUT_DOUBLE("wmo_1st_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("wmo_1st_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("wmo_1st_o3", o3t, 1);

    /* Get WMO 2nd tropopause... */
    get_tropo(4, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
              zs);
    NC_PUT_DOUBLE("wmo_2nd_z", zt, 1);
    NC_PUT_DOUBLE("wmo_2nd_p", pt, 1);
    NC_PUT_DOUBLE("wmo_2nd_t", tt, 1);
    if (h2o)
      NC_PUT_DOUBLE("wmo_2nd_q", qt, 1);
    if (o3)
      NC_PUT_DOUBLE("wmo_2nd_o3", o3t, 1);

    /* Write surface data... */
    NC_PUT_DOUBLE("ps", ps, 1);
    NC_PUT_DOUBLE("zs", zs, 1);

    /* Increment time step counter... */
    nt++;
  }

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

  /* Free... */
  free(clim);
  free(met);

  return EXIT_SUCCESS;
}

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

void get_tropo(
  int met_tropo,
```

```
  ctl_t * ctl,
  clim_t * clim,
  met_t * met,
  double *lons,
  int nx,
  double *lats,
  int ny,
  double *pt,
  double *zt,
  double *tt,
  double *qt,
  double *o3t,
  double *ps,
  double *zs) {

  INTPOL_INIT;

  ctl->met_tropo = met_tropo;
  read_met_tropo(ctl, clim, met);
#pragma omp parallel for default(shared) private(ci,cw)
  for (int ix = 0; ix < nx; ix++)
    for (int 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_2d(met, met->ps, lons[ix], lats[iy],
                          &ps[iy * nx + ix], ci, cw, 0);
      intpol_met_space_2d(met, met->zs, lons[ix], lats[iy],
                          &zs[iy * nx + ix], ci, cw, 0);
      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);
      intpol_met_space_3d(met, met->o3, pt[iy * nx + ix], lons[ix],
                          lats[iy], &o3t[iy * nx + ix], ci, cw, 0);
    }
}
```

5.51 tropo_sample.c File Reference

Sample tropopause data set.

```
#include "libtrac.h"
```

Macros

- #define NT 744

  Maximum number of time steps.

Functions

- void intpol_tropo_3d (double time0, float array0[EX][EY], double time1, float array1[EX][EY], double lons[EX], double lats[EY], int nlon, int nlat, double time, double lon, double lat, int method, double ∗var, double ∗sigma)

  3-D linear interpolation of tropopause data.
- int main (int argc, char ∗argv[ ])

5.51.1 Detailed Description

Sample tropopause data set.

Definition in file tropo_sample.c.

5.51.2 Macro Definition Documentation

5.51.2.1 NT `#define NT 744`

Maximum number of time steps.

Definition at line 32 of file tropo_sample.c.

5.51.3 Function Documentation

5.51.3.1 intpol_tropo_3d()

```
void intpol_tropo_3d (
        double time0,
        float array0[EX][EY],
        double time1,
        float array1[EX][EY],
        double lons[EX],
        double lats[EY],
        int nlon,
        int nlat,
        double time,
        double lon,
        double lat,
        int method,
        double * var,
        double * sigma )
```

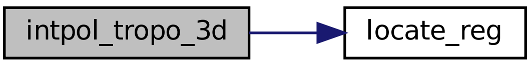
3-D linear interpolation of tropopause data.

Definition at line 279 of file tropo_sample.c.

```
                  {

  double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;

  int n = 0;

  /* 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);

  /* Calculate standard deviation... */
  *sigma = 0;
  for (int dx = 0; dx < 2; dx++)
    for (int dy = 0; dy < 2; dy++) {
      if (isfinite(array0[ix + dx][iy + dy])) {
        mean += array0[ix + dx][iy + dy];
        *sigma += SQR(array0[ix + dx][iy + dy]);
        n++;
      }
      if (isfinite(array1[ix + dx][iy + dy])) {
        mean += array1[ix + dx][iy + dy];
        *sigma += SQR(array1[ix + dx][iy + dy]);
        n++;
      }
    }
```

```
  if (n > 0)
    *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));

  /* Linear interpolation... */
  if (method == 1 && isfinite(array0[ix][iy])
      && isfinite(array0[ix][iy + 1])
      && isfinite(array0[ix + 1][iy])
      && isfinite(array0[ix + 1][iy + 1])
      && isfinite(array1[ix][iy])
      && isfinite(array1[ix][iy + 1])
      && isfinite(array1[ix + 1][iy])
      && isfinite(array1[ix + 1][iy + 1])) {

    aux00 = LIN(lons[ix], array0[ix][iy],
                lons[ix + 1], array0[ix + 1][iy], lon);
    aux01 = LIN(lons[ix], array0[ix][iy + 1],
                lons[ix + 1], array0[ix + 1][iy + 1], lon);
    aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);

    aux10 = LIN(lons[ix], array1[ix][iy],
                lons[ix + 1], array1[ix + 1][iy], lon);
    aux11 = LIN(lons[ix], array1[ix][iy + 1],
                lons[ix + 1], array1[ix + 1][iy + 1], lon);
    aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);

    *var = LIN(time0, aux0, time1, aux1, time);
  }

  /* Nearest neighbor interpolation... */
  else {
    aux00 = NN(lons[ix], array0[ix][iy],
               lons[ix + 1], array0[ix + 1][iy], lon);
    aux01 = NN(lons[ix], array0[ix][iy + 1],
               lons[ix + 1], array0[ix + 1][iy + 1], lon);
    aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);

    aux10 = NN(lons[ix], array1[ix][iy],
               lons[ix + 1], array1[ix + 1][iy], lon);
    aux11 = NN(lons[ix], array1[ix][iy + 1],
               lons[ix + 1], array1[ix + 1][iy + 1], lon);
    aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);

    *var = NN(time0, aux0, time1, aux1, time);
  }
}
```

Here is the call graph for this function:



5.51.3.2 main()

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

Definition at line 59 of file tropo_sample.c.

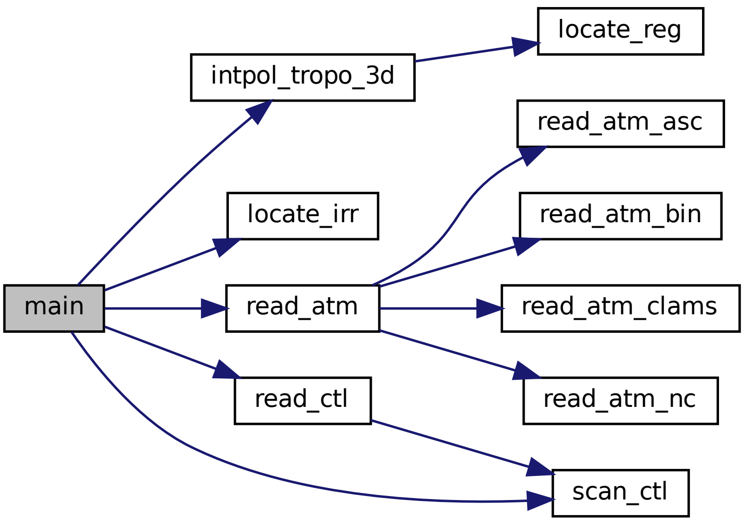
```
                      {

  ctl_t ctl;

  atm_t *atm;

  static FILE *out;

```

```
static char varname[LEN];

static double times[NT], lons[EX], lats[EY], time0, time1, z0, z0sig,
  p0, p0sig, t0, t0sig, q0, q0sig, o30, o30sig;

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], tropo_o30[EX][EY], tropo_o31[EX][EY];

static int ip, iq, it, it_old =
  -999, method, ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3,
  h2o, o3, ntime, nlon, nlat, ilon, ilat;

static size_t count[10], start[10];

/* 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);
method =
  (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);

/* Read atmospheric data... */
if (!read_atm(argv[5], &ctl, atm))
  ERRMSG("Cannot open file!");

/* Open tropopause file... */
LOG(1, "Read tropopause data: %s", argv[3]);
if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
  ERRMSG("Cannot open file!");

/* Get dimensions... */
NC_INQ_DIM("time", &ntime, 1, NT);
NC_INQ_DIM("lat", &nlat, 1, EY);
NC_INQ_DIM("lon", &nlon, 1, EX);

/* Read coordinates... */
NC_GET_DOUBLE("time", times, 1);
NC_GET_DOUBLE("lat", lats, 1);
NC_GET_DOUBLE("lon", lons, 1);

/* 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);
sprintf(varname, "%s_o3", argv[4]);
o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);

/* Set dimensions... */
count[0] = 1;
count[1] = (size_t) nlat;
count[2] = (size_t) nlon;

/* Create file... */
LOG(1, "Write tropopause sample data: %s", 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 (mean) [km]\n", 5 + ctl.nq);
fprintf(out, "# $%d = tropopause pressure (mean) [hPa]\n", 6 + ctl.nq);
fprintf(out, "# $%d = tropopause temperature (mean) [K]\n", 7 + ctl.nq);
fprintf(out, "# $%d = tropopause water vapor (mean) [ppv]\n", 8 + ctl.nq);
fprintf(out, "# $%d = tropopause ozone (mean) [ppv]\n", 9 + ctl.nq);
fprintf(out, "# $%d = tropopause height (sigma) [km]\n", 10 + ctl.nq);
fprintf(out, "# $%d = tropopause pressure (sigma) [hPa]\n", 11 + ctl.nq);
fprintf(out, "# $%d = tropopause temperature (sigma) [K]\n", 12 + ctl.nq);
fprintf(out, "# $%d = tropopause water vapor (sigma) [ppv]\n", 13 + ctl.nq);
fprintf(out, "# $%d = tropopause ozone (sigma) [ppv]\n\n", 14 + 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;
      if (o3) {
        NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[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;;
      if (o3) {
        NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o31[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o31[ilon][ilat] = GSL_NAN;;
    }
    it_old = it;

    /* Interpolate... */
    intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &z0, &z0sig);
```

```
    intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &p0, &p0sig);
    intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &t0, &t0sig);
    intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &q0, &q0sig);
    intpol_tropo_3d(time0, tropo_o30, time1, tropo_o31,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &o30, &o30sig);

    /* 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 %g %g %g %g %g %g\n",
            z0, p0, t0, q0, o30, z0sig, p0sig, t0sig, q0sig, o30sig);
  }

  /* Close files... */
  fclose(out);
  NC(nc_close(ncid));

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.52 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) 2013-2023 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

#define NT 744

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

void intpol_tropo_3d(
  double time0,
  float array0[EX][EY],
  double time1,
  float array1[EX][EY],
  double lons[EX],
  double lats[EY],
  int nlon,
  int nlat,
  double time,
  double lon,
  double lat,
  int method,
  double *var,
  double *sigma);

/* ------------------------------------------------------------
   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, z0sig,
    p0, p0sig, t0, t0sig, q0, q0sig, o30, o30sig;

  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], tropo_o30[EX][EY], tropo_o31[EX][EY];

  static int ip, iq, it, it_old =
    -999, method, ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3,
    h2o, o3, ntime, nlon, nlat, ilon, ilat;

  static size_t count[10], start[10];

  /* 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);
  method =
    (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);

  /* Read atmospheric data... */
  if (!read_atm(argv[5], &ctl, atm))
    ERRMSG("Cannot open file!");

  /* Open tropopause file... */
  LOG(1, "Read tropopause data: %s", argv[3]);
  if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
    ERRMSG("Cannot open file!");
```

```
/* Get dimensions... */
NC_INQ_DIM("time", &ntime, 1, NT);
NC_INQ_DIM("lat", &nlat, 1, EY);
NC_INQ_DIM("lon", &nlon, 1, EX);

/* Read coordinates... */
NC_GET_DOUBLE("time", times, 1);
NC_GET_DOUBLE("lat", lats, 1);
NC_GET_DOUBLE("lon", lons, 1);

/* 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);
sprintf(varname, "%s_o3", argv[4]);
o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);

/* Set dimensions... */
count[0] = 1;
count[1] = (size_t) nlat;
count[2] = (size_t) nlon;

/* Create file... */
LOG(1, "Write tropopause sample data: %s", 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 (mean) [km]\n", 5 + ctl.nq);
fprintf(out, "# $%d = tropopause pressure (mean) [hPa]\n", 6 + ctl.nq);
fprintf(out, "# $%d = tropopause temperature (mean) [K]\n", 7 + ctl.nq);
fprintf(out, "# $%d = tropopause water vapor (mean) [ppv]\n", 8 + ctl.nq);
fprintf(out, "# $%d = tropopause ozone (mean) [ppv]\n", 9 + ctl.nq);
fprintf(out, "# $%d = tropopause height (sigma) [km]\n", 10 + ctl.nq);
fprintf(out, "# $%d = tropopause pressure (sigma) [hPa]\n", 11 + ctl.nq);
fprintf(out, "# $%d = tropopause temperature (sigma) [K]\n", 12 + ctl.nq);
fprintf(out, "# $%d = tropopause water vapor (sigma) [ppv]\n", 13 + ctl.nq);
fprintf(out, "# $%d = tropopause ozone (sigma) [ppv]\n\n", 14 + 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;
      if (o3) {
        NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[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;;
      if (o3) {
        NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o31[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o31[ilon][ilat] = GSL_NAN;;
    }
    it_old = it;

    /* Interpolate... */
    intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &z0, &z0sig);
    intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &p0, &p0sig);
    intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &t0, &t0sig);
    intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &q0, &q0sig);
    intpol_tropo_3d(time0, tropo_o30, time1, tropo_o31,
                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
                    atm->lat[ip], method, &o30, &o30sig);

    /* 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 %g %g %g %g %g %g\n",
            z0, p0, t0, q0, o30, z0sig, p0sig, t0sig, q0sig, o30sig);
  }

  /* Close files... */
  fclose(out);
  NC(nc_close(ncid));

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

  return EXIT_SUCCESS;
}

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

```
void intpol_tropo_3d(
  double time0,
  float array0[EX][EY],
  double time1,
  float array1[EX][EY],
  double lons[EX],
  double lats[EY],
  int nlon,
  int nlat,
  double time,
  double lon,
  double lat,
  int method,
  double *var,
  double *sigma) {

  double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;

  int n = 0;

  /* 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);

  /* Calculate standard deviation... */
  *sigma = 0;
  for (int dx = 0; dx < 2; dx++)
    for (int dy = 0; dy < 2; dy++) {
      if (isfinite(array0[ix + dx][iy + dy])) {
        mean += array0[ix + dx][iy + dy];
        *sigma += SQR(array0[ix + dx][iy + dy]);
        n++;
      }
      if (isfinite(array1[ix + dx][iy + dy])) {
        mean += array1[ix + dx][iy + dy];
        *sigma += SQR(array1[ix + dx][iy + dy]);
        n++;
      }
    }
  if (n > 0)
    *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));

  /* Linear interpolation... */
  if (method == 1 && isfinite(array0[ix][iy])
      && isfinite(array0[ix][iy + 1])
      && isfinite(array0[ix + 1][iy])
      && isfinite(array0[ix + 1][iy + 1])
      && isfinite(array1[ix][iy])
      && isfinite(array1[ix][iy + 1])
      && isfinite(array1[ix + 1][iy])
      && isfinite(array1[ix + 1][iy + 1])) {

    aux00 = LIN(lons[ix], array0[ix][iy],
                lons[ix + 1], array0[ix + 1][iy], lon);
    aux01 = LIN(lons[ix], array0[ix][iy + 1],
                lons[ix + 1], array0[ix + 1][iy + 1], lon);
    aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);

    aux10 = LIN(lons[ix], array1[ix][iy],
                lons[ix + 1], array1[ix + 1][iy], lon);
    aux11 = LIN(lons[ix], array1[ix][iy + 1],
                lons[ix + 1], array1[ix + 1][iy + 1], lon);
    aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);

    *var = LIN(time0, aux0, time1, aux1, time);
  }

  /* Nearest neighbor interpolation... */
  else {
    aux00 = NN(lons[ix], array0[ix][iy],
               lons[ix + 1], array0[ix + 1][iy], lon);
    aux01 = NN(lons[ix], array0[ix][iy + 1],
               lons[ix + 1], array0[ix + 1][iy + 1], lon);
    aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);

    aux10 = NN(lons[ix], array1[ix][iy],
               lons[ix + 1], array1[ix + 1][iy], lon);
    aux11 = NN(lons[ix], array1[ix][iy + 1],
               lons[ix + 1], array1[ix + 1][iy + 1], lon);
    aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
```

```
    *var = NN(time0, aux0, time1, aux1, time);
  }
}
```

5.53 tropo_zm.c File Reference

Extract zonal mean of tropopause data set.

```
#include "libtrac.h"
```

Macros

- #define NT 744

  Maximum number of time steps.

Functions

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

5.53.1 Detailed Description

Extract zonal mean of tropopause data set.

Definition in file tropo_zm.c.

5.53.2 Macro Definition Documentation

5.53.2.1 NT `#define NT 744`

Maximum number of time steps.

Definition at line 32 of file tropo_zm.c.

5.53.3 Function Documentation

5.53.3.1 main()

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

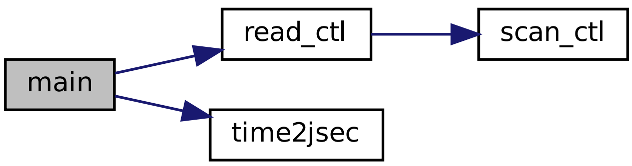
Definition at line 38 of file tropo_zm.c.

```
                {

  ctl_t ctl;

  static FILE *out;

  static char tstr[LEN], varname[LEN];

  static double time0, lons[EX], lats[EY], zm[EY], zs[EY], pm[EY],
    ps[EY], tm[EY], ts[EY], qm[EY], qs[EY], o3m[EY], o3s[EY];

  static float help[EX * EY], tropo_z0[EX][EY], tropo_p0[EX][EY],
    tropo_t0[EX][EY], tropo_q0[EX][EY], tropo_o30[EX][EY];

  static int ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3, h2o,
    o3, n[EY], nt[EY], year, mon, day, init, ntime, nlon, nlat, ilon, ilat;

  static size_t count[10], start[10];

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <zm.tab> <var> <tropo.nc>");

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

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

    /* Open tropopause file... */
    LOG(1, "Read tropopause data: %s", argv[iarg]);
    if (nc_open(argv[iarg], NC_NOWRITE, &ncid) != NC_NOERR)
      ERRMSG("Cannot open file!");

    /* Get dimensions... */
    NC_INQ_DIM("time", &ntime, 1, NT);
    NC_INQ_DIM("lat", &nlat, 1, EY);
    NC_INQ_DIM("lon", &nlon, 1, EX);

    /* Read coordinates... */
    NC_GET_DOUBLE("lat", lats, 1);
    NC_GET_DOUBLE("lon", lons, 1);

    /* Get variable indices... */
    sprintf(varname, "%s_z", argv[3]);
    NC(nc_inq_varid(ncid, varname, &varid_z));
    sprintf(varname, "%s_p", argv[3]);
    NC(nc_inq_varid(ncid, varname, &varid_p));
    sprintf(varname, "%s_t", argv[3]);
    NC(nc_inq_varid(ncid, varname, &varid_t));
    sprintf(varname, "%s_q", argv[3]);
    h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
    sprintf(varname, "%s_o3", argv[3]);
    o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);

    /* Set dimensions... */
    count[0] = 1;
    count[1] = (size_t) nlat;
    count[2] = (size_t) nlon;

    /* Loop over time steps... */
    for (int it = 0; it < ntime; it++) {

      /* Get time from filename... */
      if (!init) {
        init = 1;
        size_t len = strlen(argv[iarg]);
        sprintf(tstr, "%.4s", &argv[iarg][len - 13]);
        year = atoi(tstr);
        sprintf(tstr, "%.2s", &argv[iarg][len - 8]);
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &argv[iarg][len - 5]);
        day = atoi(tstr);
        time2jsec(year, mon, day, 0, 0, 0, 0, &time0);
      }

      /* Read data... */
      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;
      if (o3) {
        NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[ilon][ilat] = GSL_NAN;

      /* Averaging... */
      for (ilat = 0; ilat < nlat; ilat++)
        for (ilon = 0; ilon < nlon; ilon++) {
          nt[ilat]++;
          if (isfinite(tropo_z0[ilon][ilat])
              && isfinite(tropo_p0[ilon][ilat])
              && isfinite(tropo_t0[ilon][ilat])
              && (!h2o || isfinite(tropo_q0[ilon][ilat]))
              && (!o3 || isfinite(tropo_o30[ilon][ilat]))) {
            zm[ilat] += tropo_z0[ilon][ilat];
            zs[ilat] += SQR(tropo_z0[ilon][ilat]);
            pm[ilat] += tropo_p0[ilon][ilat];
            ps[ilat] += SQR(tropo_p0[ilon][ilat]);
            tm[ilat] += tropo_t0[ilon][ilat];
            ts[ilat] += SQR(tropo_t0[ilon][ilat]);
            qm[ilat] += tropo_q0[ilon][ilat];
            qs[ilat] += SQR(tropo_q0[ilon][ilat]);
            o3m[ilat] += tropo_o30[ilon][ilat];
            o3s[ilat] += SQR(tropo_o30[ilon][ilat]);
            n[ilat]++;
          }
        }
    }

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

  /* Normalize... */
  for (ilat = 0; ilat < nlat; ilat++)
    if (n[ilat] > 0) {
      zm[ilat] /= n[ilat];
      pm[ilat] /= n[ilat];
      tm[ilat] /= n[ilat];
      qm[ilat] /= n[ilat];
      o3m[ilat] /= n[ilat];
      double aux = zs[ilat] / n[ilat] - SQR(zm[ilat]);
      zs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = ps[ilat] / n[ilat] - SQR(pm[ilat]);
      ps[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = ts[ilat] / n[ilat] - SQR(tm[ilat]);
      ts[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = qs[ilat] / n[ilat] - SQR(qm[ilat]);
      qs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = o3s[ilat] / n[ilat] - SQR(o3m[ilat]);
      o3s[ilat] = aux > 0 ? sqrt(aux) : 0.0;
    }

  /* Create file... */
  LOG(1, "Write tropopause zonal mean data: %s", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = latitude [deg]\n"
          "# $3 = tropopause height (mean) [km]\n"
```

```
          "# $4 = tropopause pressure (mean) [hPa]\n"
          "# $5 = tropopause temperature (mean) [K]\n"
          "# $6 = tropopause water vapor (mean) [ppv]\n"
          "# $7 = tropopause ozone (mean) [ppv]\n"
          "# $8 = tropopause height (sigma) [km]\n"
          "# $9 = tropopause pressure (sigma) [hPa]\n"
          "# $10 = tropopause temperature (sigma) [K]\n"
          "# $11 = tropopause water vapor (sigma) [ppv]\n"
          "# $12 = tropopause ozone (sigma) [ppv]\n"
          "# $13 = number of data points\n"
          "# $14 = occurrence frequency [%%]\n\n");

  /* Write output... */
  for (ilat = 0; ilat < nlat; ilat++)
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %d %g\n",
            time0, lats[ilat], zm[ilat], pm[ilat], tm[ilat], qm[ilat],
            o3m[ilat], zs[ilat], ps[ilat], ts[ilat], qs[ilat], o3s[ilat],
            n[ilat], 100. * n[ilat] / nt[ilat]);

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.54 tropo_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) 2013-2023 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

#define NT 744

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

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

```

  ctl_t ctl;

  static FILE *out;

  static char tstr[LEN], varname[LEN];

  static double time0, lons[EX], lats[EY], zm[EY], zs[EY], pm[EY],
    ps[EY], tm[EY], ts[EY], qm[EY], qs[EY], o3m[EY], o3s[EY];

  static float help[EX * EY], tropo_z0[EX][EY], tropo_p0[EX][EY],
    tropo_t0[EX][EY], tropo_q0[EX][EY], tropo_o30[EX][EY];

  static int ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3, h2o,
    o3, n[EY], nt[EY], year, mon, day, init, ntime, nlon, nlat, ilon, ilat;

  static size_t count[10], start[10];

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <zm.tab> <var> <tropo.nc>");

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

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

    /* Open tropopause file... */
    LOG(1, "Read tropopause data: %s", argv[iarg]);
    if (nc_open(argv[iarg], NC_NOWRITE, &ncid) != NC_NOERR)
      ERRMSG("Cannot open file!");

    /* Get dimensions... */
    NC_INQ_DIM("time", &ntime, 1, NT);
    NC_INQ_DIM("lat", &nlat, 1, EY);
    NC_INQ_DIM("lon", &nlon, 1, EX);

    /* Read coordinates... */
    NC_GET_DOUBLE("lat", lats, 1);
    NC_GET_DOUBLE("lon", lons, 1);

    /* Get variable indices... */
    sprintf(varname, "%s_z", argv[3]);
    NC(nc_inq_varid(ncid, varname, &varid_z));
    sprintf(varname, "%s_p", argv[3]);
    NC(nc_inq_varid(ncid, varname, &varid_p));
    sprintf(varname, "%s_t", argv[3]);
    NC(nc_inq_varid(ncid, varname, &varid_t));
    sprintf(varname, "%s_q", argv[3]);
    h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
    sprintf(varname, "%s_o3", argv[3]);
    o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);

    /* Set dimensions... */
    count[0] = 1;
    count[1] = (size_t) nlat;
    count[2] = (size_t) nlon;

    /* Loop over time steps... */
    for (int it = 0; it < ntime; it++) {

      /* Get time from filename... */
      if (!init) {
        init = 1;
        size_t len = strlen(argv[iarg]);
        sprintf(tstr, "%.4s", &argv[iarg][len - 13]);
        year = atoi(tstr);
        sprintf(tstr, "%.2s", &argv[iarg][len - 8]);
        mon = atoi(tstr);
        sprintf(tstr, "%.2s", &argv[iarg][len - 5]);
        day = atoi(tstr);
        time2jsec(year, mon, day, 0, 0, 0, 0, &time0);
      }

      /* Read data... */
      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;
      if (o3) {
        NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
      } else
        for (ilon = 0; ilon < nlon; ilon++)
          for (ilat = 0; ilat < nlat; ilat++)
            tropo_o30[ilon][ilat] = GSL_NAN;

      /* Averaging... */
      for (ilat = 0; ilat < nlat; ilat++)
        for (ilon = 0; ilon < nlon; ilon++) {
          nt[ilat]++;
          if (isfinite(tropo_z0[ilon][ilat])
              && isfinite(tropo_p0[ilon][ilat])
              && isfinite(tropo_t0[ilon][ilat])
              && (!h2o || isfinite(tropo_q0[ilon][ilat]))
              && (!o3 || isfinite(tropo_o30[ilon][ilat]))) {
            zm[ilat] += tropo_z0[ilon][ilat];
            zs[ilat] += SQR(tropo_z0[ilon][ilat]);
            pm[ilat] += tropo_p0[ilon][ilat];
            ps[ilat] += SQR(tropo_p0[ilon][ilat]);
            tm[ilat] += tropo_t0[ilon][ilat];
            ts[ilat] += SQR(tropo_t0[ilon][ilat]);
            qm[ilat] += tropo_q0[ilon][ilat];
            qs[ilat] += SQR(tropo_q0[ilon][ilat]);
            o3m[ilat] += tropo_o30[ilon][ilat];
            o3s[ilat] += SQR(tropo_o30[ilon][ilat]);
            n[ilat]++;
          }
        }
    }

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

  /* Normalize... */
  for (ilat = 0; ilat < nlat; ilat++)
    if (n[ilat] > 0) {
      zm[ilat] /= n[ilat];
      pm[ilat] /= n[ilat];
      tm[ilat] /= n[ilat];
      qm[ilat] /= n[ilat];
      o3m[ilat] /= n[ilat];
      double aux = zs[ilat] / n[ilat] - SQR(zm[ilat]);
      zs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = ps[ilat] / n[ilat] - SQR(pm[ilat]);
      ps[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = ts[ilat] / n[ilat] - SQR(tm[ilat]);
      ts[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = qs[ilat] / n[ilat] - SQR(qm[ilat]);
      qs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
      aux = o3s[ilat] / n[ilat] - SQR(o3m[ilat]);
      o3s[ilat] = aux > 0 ? sqrt(aux) : 0.0;
    }

  /* Create file... */
  LOG(1, "Write tropopause zonal mean data: %s", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = latitude [deg]\n"
          "# $3 = tropopause height (mean) [km]\n"
          "# $4 = tropopause pressure (mean) [hPa]\n"
          "# $5 = tropopause temperature (mean) [K]\n"
          "# $6 = tropopause water vapor (mean) [ppv]\n"
          "# $7 = tropopause ozone (mean) [ppv]\n"
          "# $8 = tropopause height (sigma) [km]\n"
          "# $9 = tropopause pressure (sigma) [hPa]\n"
          "# $10 = tropopause temperature (sigma) [K]\n"
          "# $11 = tropopause water vapor (sigma) [ppv]\n"
```

```
        "# $12 = tropopause ozone (sigma) [ppv]\n"
        "# $13 = number of data points\n"
        "# $14 = occurrence frequency [%%]\n\n");

  /* Write output... */
  for (ilat = 0; ilat < nlat; ilat++)
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %d %g\n",
            time0, lats[ilat], zm[ilat], pm[ilat], tm[ilat], qm[ilat],
            o3m[ilat], zs[ilat], ps[ilat], ts[ilat], qs[ilat], o3s[ilat],
            n[ilat], 100. * n[ilat] / nt[ilat]);

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

  return EXIT_SUCCESS;
}
```

5.55 wind.c File Reference

Create meteorological data files with synthetic wind fields.

```
#include "libtrac.h"
```

Functions

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

5.55.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.55.2 Function Documentation

5.55.2.1 main()

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

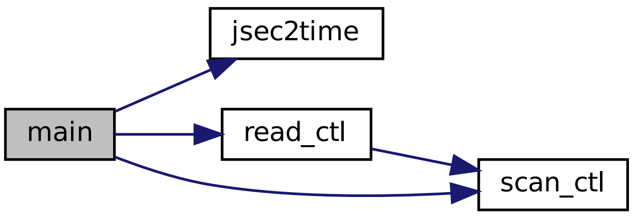
Definition at line 31 of file wind.c.

```
                 {

  ctl_t ctl;

  static char filename[LEN];

  static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
    u0, u1, w0, alpha;

  static float *dataT, *dataU, *dataV, *dataW;

  static int ncid, varid, dims[4], idx, ix, iy, iz, nx, ny, nz,
    year, mon, day, hour, min, sec;

  static size_t start[4], count[4];

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

```
  ALLOC(dataU, float,
        EP * EY * EX);
  ALLOC(dataV, float,
        EP * EY * EX);
  ALLOC(dataW, float,
        EP * EY * EX);

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

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
  nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
  ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
  nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
  z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
  u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
  u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
  w0 = scan_ctl(argv[1], argc, argv, "WIND_W0", -1, "0", NULL);
  alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);

  /* Check dimensions... */
  if (nx < 1 || nx > EX)
    ERRMSG("Set 1 <= NX <= MAX!");
  if (ny < 1 || ny > EY)
    ERRMSG("Set 1 <= NY <= MAX!");
  if (nz < 1 || nz > EP)
    ERRMSG("Set 1 <= NZ <= MAX!");

  /* Get time... */
  jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
  t0 = year * 10000. + mon * 100. + day + hour / 24.;

  /* Set filename... */
  sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);

  /* Create netCDF file... */
  NC(nc_create(filename, NC_CLOBBER, &ncid));

  /* Create dimensions... */
  NC(nc_def_dim(ncid, "time", 1, &dims[0]));
  NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
  NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
  NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));

  /* Create variables... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time", "day as %Y%m%d.%f");
  NC_DEF_VAR("lev", NC_DOUBLE, 1, &dims[1], "air_pressure", "Pa");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[2], "latitude", "degrees_north");
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[3], "longitude", "degrees_east");
  NC_DEF_VAR("T", NC_FLOAT, 4, &dims[0], "Temperature", "K");
  NC_DEF_VAR("U", NC_FLOAT, 4, &dims[0], "zonal wind", "m s**-1");
  NC_DEF_VAR("V", NC_FLOAT, 4, &dims[0], "meridional wind", "m s**-1");
  NC_DEF_VAR("W", NC_FLOAT, 4, &dims[0], "vertical velocity", "Pa s**-1");

  /* End definition... */
  NC(nc_enddef(ncid));

  /* Set coordinates... */
  for (ix = 0; ix < nx; ix++)
    dataLon[ix] = 360.0 / nx * (double) ix;
  for (iy = 0; iy < ny; iy++)
    dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
  for (iz = 0; iz < nz; iz++)
    dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));

  /* Write coordinates... */
  NC_PUT_DOUBLE("time", &t0, 0);
  NC_PUT_DOUBLE("lev", dataZ, 0);
  NC_PUT_DOUBLE("lat", dataLat, 0);
  NC_PUT_DOUBLE("lon", dataLon, 0);

  /* Create wind fields (Williamson et al., 1992)... */
  for (ix = 0; ix < nx; ix++)
    for (iy = 0; iy < ny; iy++)
      for (iz = 0; iz < nz; iz++) {
        idx = (iz * ny + iy) * nx + ix;
        dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
                              * (cos(dataLat[iy] * M_PI / 180.0)
                                 * cos(alpha * M_PI / 180.0)
                                 + sin(dataLat[iy] * M_PI / 180.0)
                                 * cos(dataLon[ix] * M_PI / 180.0)
                                 * sin(alpha * M_PI / 180.0)));
        dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
```

```
                                   * sin(dataLon[ix] * M_PI / 180.0)
                                   * sin(alpha * M_PI / 180.0));
        dataW[idx] = (float) DZ2DP(1e-3 * w0, dataZ[iz]);
      }

  /* Write data... */
  NC_PUT_FLOAT("T", dataT, 0);
  NC_PUT_FLOAT("U", dataU, 0);
  NC_PUT_FLOAT("V", dataV, 0);
  NC_PUT_FLOAT("W", dataW, 0);

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

  /* Free... */
  free(dataT);
  free(dataU);
  free(dataV);
  free(dataW);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.56 wind.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) 2013-2021 Forschungszentrum Juelich GmbH
*/

#include "libtrac.h"

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

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

  ctl_t ctl;

  static char filename[LEN];

  static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
```

```
    u0, u1, w0, alpha;

  static float *dataT, *dataU, *dataV, *dataW;

  static int ncid, varid, dims[4], idx, ix, iy, iz, nx, ny, nz,
    year, mon, day, hour, min, sec;

  static size_t start[4], count[4];

  /* Allocate... */
  ALLOC(dataT, float,
        EP * EY * EX);
  ALLOC(dataU, float,
        EP * EY * EX);
  ALLOC(dataV, float,
        EP * EY * EX);
  ALLOC(dataW, float,
        EP * EY * EX);

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

  /* Read control parameters... */
  read_ctl(argv[1], argc, argv, &ctl);
  t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
  nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
  ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
  nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
  z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
  z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
  u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
  u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
  w0 = scan_ctl(argv[1], argc, argv, "WIND_W0", -1, "0", NULL);
  alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);

  /* Check dimensions... */
  if (nx < 1 || nx > EX)
    ERRMSG("Set 1 <= NX <= MAX!");
  if (ny < 1 || ny > EY)
    ERRMSG("Set 1 <= NY <= MAX!");
  if (nz < 1 || nz > EP)
    ERRMSG("Set 1 <= NZ <= MAX!");

  /* Get time... */
  jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
  t0 = year * 10000. + mon * 100. + day + hour / 24.;

  /* Set filename... */
  sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);

  /* Create netCDF file... */
  NC(nc_create(filename, NC_CLOBBER, &ncid));

  /* Create dimensions... */
  NC(nc_def_dim(ncid, "time", 1, &dims[0]));
  NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
  NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
  NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));

  /* Create variables... */
  NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time", "day as %Y%m%d.%f");
  NC_DEF_VAR("lev", NC_DOUBLE, 1, &dims[1], "air_pressure", "Pa");
  NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[2], "latitude", "degrees_north");
  NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[3], "longitude", "degrees_east");
  NC_DEF_VAR("T", NC_FLOAT, 4, &dims[0], "Temperature", "K");
  NC_DEF_VAR("U", NC_FLOAT, 4, &dims[0], "zonal wind", "m s**-1");
  NC_DEF_VAR("V", NC_FLOAT, 4, &dims[0], "meridional wind", "m s**-1");
  NC_DEF_VAR("W", NC_FLOAT, 4, &dims[0], "vertical velocity", "Pa s**-1");

  /* End definition... */
  NC(nc_enddef(ncid));

  /* Set coordinates... */
  for (ix = 0; ix < nx; ix++)
    dataLon[ix] = 360.0 / nx * (double) ix;
  for (iy = 0; iy < ny; iy++)
    dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
  for (iz = 0; iz < nz; iz++)
    dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));

  /* Write coordinates... */
  NC_PUT_DOUBLE("time", &t0, 0);
  NC_PUT_DOUBLE("lev", dataZ, 0);
  NC_PUT_DOUBLE("lat", dataLat, 0);
  NC_PUT_DOUBLE("lon", dataLon, 0);

```

```
  /* Create wind fields (Williamson et al., 1992)... */
  for (ix = 0; ix < nx; ix++)
    for (iy = 0; iy < ny; iy++)
      for (iz = 0; iz < nz; iz++) {
        idx = (iz * ny + iy) * nx + ix;
        dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
                               * (cos(dataLat[iy] * M_PI / 180.0)
                                  * cos(alpha * M_PI / 180.0)
                                  + sin(dataLat[iy] * M_PI / 180.0)
                                  * cos(dataLon[ix] * M_PI / 180.0)
                                  * sin(alpha * M_PI / 180.0)));
        dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
                               * sin(dataLon[ix] * M_PI / 180.0)
                               * sin(alpha * M_PI / 180.0));
        dataW[idx] = (float) DZ2DP(1e-3 * w0, dataZ[iz]);
      }

  /* Write data... */
  NC_PUT_FLOAT("T", dataT, 0);
  NC_PUT_FLOAT("U", dataU, 0);
  NC_PUT_FLOAT("V", dataV, 0);
  NC_PUT_FLOAT("W", dataW, 0);

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

  /* Free... */
  free(dataT);
  free(dataU);
  free(dataV);
  free(dataW);

  return EXIT_SUCCESS;
}
```

Index