

MPTRAC

Generated by Doxygen 1.8.17

1 Main Page	1
2 Data Structure Index	1
2.1 Data Structures	1
3 File Index	1
3.1 File List	1
4 Data Structure Documentation	2
4.1 atm_t Struct Reference	2
4.1.1 Detailed Description	3
4.1.2 Field Documentation	3
4.2 cache_t Struct Reference	4
4.2.1 Detailed Description	4
4.2.2 Field Documentation	4
4.3 ctl_t Struct Reference	6
4.3.1 Detailed Description	13
4.3.2 Field Documentation	13
4.4 met_t Struct Reference	42
4.4.1 Detailed Description	44
4.4.2 Field Documentation	44
5 File Documentation	50
5.1 atm_conv.c File Reference	50
5.1.1 Detailed Description	50
5.1.2 Function Documentation	51
5.2 atm_conv.c	51
5.3 atm_dist.c File Reference	52
5.3.1 Detailed Description	52
5.3.2 Function Documentation	52
5.4 atm_dist.c	57
5.5 atm_init.c File Reference	62
5.5.1 Detailed Description	62
5.5.2 Function Documentation	62
5.6 atm_init.c	64
5.7 atm_select.c File Reference	65
5.7.1 Detailed Description	66
5.7.2 Function Documentation	66
5.8 atm_select.c	68
5.9 atm_split.c File Reference	70
5.9.1 Detailed Description	70
5.9.2 Function Documentation	70
5.10 atm_split.c	72
5.11 atm_stat.c File Reference	74

5.11.1 Detailed Description	74
5.11.2 Function Documentation	74
5.12 atm_stat.c	77
5.13 day2doy.c File Reference	80
5.13.1 Detailed Description	80
5.13.2 Function Documentation	80
5.14 day2doy.c	81
5.15 doy2day.c File Reference	82
5.15.1 Detailed Description	82
5.15.2 Function Documentation	82
5.16 doy2day.c	83
5.17 jsec2time.c File Reference	83
5.17.1 Detailed Description	83
5.17.2 Function Documentation	83
5.18 jsec2time.c	84
5.19 libtrac.c File Reference	85
5.19.1 Detailed Description	87
5.19.2 Function Documentation	87
5.20 libtrac.c	151
5.21 libtrac.h File Reference	217
5.21.1 Detailed Description	223
5.21.2 Macro Definition Documentation	223
5.21.3 Function Documentation	239
5.22 libtrac.h	303
5.23 met_lapse.c File Reference	318
5.23.1 Detailed Description	319
5.23.2 Macro Definition Documentation	319
5.23.3 Function Documentation	319
5.24 met_lapse.c	322
5.25 met_map.c File Reference	325
5.25.1 Detailed Description	325
5.25.2 Macro Definition Documentation	325
5.25.3 Function Documentation	326
5.26 met_map.c	329
5.27 met_prof.c File Reference	332
5.27.1 Detailed Description	333
5.27.2 Macro Definition Documentation	333
5.27.3 Function Documentation	333
5.28 met_prof.c	336
5.29 met_sample.c File Reference	339
5.29.1 Detailed Description	340
5.29.2 Function Documentation	340

5.30 met_sample.c	342
5.31 met_spec.c File Reference	344
5.31.1 Detailed Description	345
5.31.2 Macro Definition Documentation	345
5.31.3 Function Documentation	345
5.32 met_spec.c	347
5.33 met_subgrid.c File Reference	349
5.33.1 Detailed Description	350
5.33.2 Function Documentation	350
5.34 met_subgrid.c	352
5.35 met_zm.c File Reference	355
5.35.1 Detailed Description	355
5.35.2 Macro Definition Documentation	355
5.35.3 Function Documentation	355
5.36 met_zm.c	359
5.37 sedi.c File Reference	362
5.37.1 Detailed Description	363
5.37.2 Function Documentation	363
5.38 sedi.c	364
5.39 time2jsec.c File Reference	364
5.39.1 Detailed Description	364
5.39.2 Function Documentation	365
5.40 time2jsec.c	365
5.41 tnat.c File Reference	366
5.41.1 Detailed Description	366
5.41.2 Function Documentation	366
5.42 tnat.c	367
5.43 trac.c File Reference	368
5.43.1 Detailed Description	369
5.43.2 Function Documentation	369
5.44 trac.c	393
5.45 tropo.c File Reference	412
5.45.1 Detailed Description	412
5.45.2 Function Documentation	412
5.46 tropo.c	417
5.47 tropo_sample.c File Reference	421
5.47.1 Detailed Description	421
5.47.2 Macro Definition Documentation	421
5.47.3 Function Documentation	421
5.48 tropo_sample.c	426
5.49 wind.c File Reference	430
5.49.1 Detailed Description	430

5.49.2 Function Documentation	430
5.50 wind.c	433
Index	437

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:

atm_t	
Atmospheric data	2
cache_t	
Cache data	4
ctl_t	
Control parameters	6
met_t	
Meteorological data	42

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

atm_conv.c	50
atm_dist.c	52
atm_init.c	62
atm_select.c	65
atm_split.c	70
atm_stat.c	74

day2doy.c	80
doy2day.c	82
jsec2time.c	83
libtrac.c	85
libtrac.h	217
met_lapse.c	318
met_map.c	325
met_prof.c	332
met_sample.c	339
met_spec.c	344
met_subgrid.c	349
met_zm.c	355
sedi.c	362
time2jsec.c	364
tnat.c	366
trac.c	368
tropo.c	412
tropo_sample.c	421
wind.c	430

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 [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 1088 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 1091 of file `libtrac.h`.

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

Time [s].

Definition at line 1094 of file `libtrac.h`.

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

Pressure [hPa].

Definition at line 1097 of file `libtrac.h`.

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

Longitude [deg].

Definition at line 1100 of file `libtrac.h`.

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

Latitude [deg].

Definition at line 1103 of file [libtrac.h](#).

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

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

Definition at line 1106 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 tsig` [EX][EY][EP]
Cache for reference time of wind standard deviations.
- `float uvwsig` [EX][EY][EP][3]
Cache for wind standard deviations.
- `float uvwp` [NP][3]
Wind perturbations [m/s].
- `double iso_var` [NP]
Isosurface variables.
- `double iso_ps` [NP]
Isosurface balloon pressure [hPa].
- `double iso_ts` [NP]
Isosurface balloon time [s].
- `int iso_n`
Isosurface balloon number of data points.

4.2.1 Detailed Description

Cache data.

Definition at line 1111 of file [libtrac.h](#).

4.2.2 Field Documentation

4.2.2.1 `tsig` `double cache_t::tsig`[\[EX\]](#)[\[EY\]](#)[\[EP\]](#)

Cache for reference time of wind standard deviations.

Definition at line [1114](#) of file [libtrac.h](#).

4.2.2.2 `uvwsig` `float cache_t::uvwsig`[\[EX\]](#)[\[EY\]](#)[\[EP\]](#)[\[3\]](#)

Cache for wind standard deviations.

Definition at line [1117](#) of file [libtrac.h](#).

4.2.2.3 `uvwp` `float cache_t::uvwp`[\[NP\]](#)[\[3\]](#)

Wind perturbations [m/s].

Definition at line [1120](#) of file [libtrac.h](#).

4.2.2.4 `iso_var` `double cache_t::iso_var`[\[NP\]](#)

Isosurface variables.

Definition at line [1123](#) of file [libtrac.h](#).

4.2.2.5 `iso_ps` `double cache_t::iso_ps`[\[NP\]](#)

Isosurface balloon pressure [hPa].

Definition at line [1126](#) of file [libtrac.h](#).

4.2.2.6 `iso_ts` `double cache_t::iso_ts`[\[NP\]](#)

Isosurface balloon time [s].

Definition at line [1129](#) of file [libtrac.h](#).

4.2.2.7 iso_n `int cache_t::iso_n`

Isosurface balloon number of data points.

Definition at line 1132 of file [libtrac.h](#).

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

- [libtrac.h](#)

4.3 ctl_t Struct Reference

Control parameters.

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

Data Fields

- `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_unit [NQ][LEN]`
Quantity units.
- `char qnt_format [NQ][LEN]`
Quantity output format.
- `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_rho`
Quantity array index for particle density.
- `int qnt_r`
Quantity array index for particle radius.
- `int qnt_ps`
Quantity array index for surface pressure.
- `int qnt_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_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`

- `qnt_psat`
 Quantity array index for hydroxyl number concentrations.
- `qnt_psice`
 Quantity array index for saturation pressure over water.
- `qnt_pw`
 Quantity array index for saturation pressure over ice.
- `qnt_sh`
 Quantity array index for partial water vapor pressure.
- `qnt_rh`
 Quantity array index for specific humidity.
- `qnt_rhice`
 Quantity array index for relative humidity over water.
- `qnt_rhice`
 Quantity array index for relative humidity over ice.
- `qnt_theta`
 Quantity array index for potential temperature.
- `qnt_zeta`
 Quantity array index for zeta vertical coordinate.
- `qnt_tvirt`
 Quantity array index for virtual temperature.
- `qnt_lapse`
 Quantity array index for lapse rate.
- `qnt_vh`
 Quantity array index for horizontal wind.
- `qnt_vz`
 Quantity array index for vertical velocity.
- `qnt_pv`
 Quantity array index for potential vorticity.
- `qnt_tdew`
 Quantity array index for dew point temperature.
- `qnt_tice`
 Quantity array index for T_{ice} .
- `qnt_tsts`
 Quantity array index for T_{STS} .
- `qnt_tnat`
 Quantity array index for T_{NAT} .
- `direction`
 Direction flag (1=forward calculation, -1=backward calculation).
- `t_start`
 Start time of simulation [s].
- `t_stop`
 Stop time of simulation [s].
- `dt_mod`
 Time step of simulation [s].
- `metbase` [LEN]
 Basename for meteorological data.
- `dt_met`
 Time step of meteorological data [s].
- `met_dx`
 Stride for longitudes.
- `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_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 width (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 width (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).
- double `met_cloud`
Cloud data (0=none, 1=LWC+IWC, 2=RWC+SWC, 3=all).
- 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).
- int `isosurf`
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
- char `balloon` [LEN]
Balloon position filename.
- int `advect`
Advection scheme (0=midpoint, 1=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_wmax`
- Maximum vertical velocity for convection module [m/s].*

 - double `conv_wcape`
- Limit vertical velocity based on CAPE (0=no, 1=yes).*

 - double `conv_dt`
- Time interval for convection module [s].*

 - 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_vmr`
- Boundary conditions volume mixing ratio [ppv].*

 - 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 delta to surface pressure [hPa].*

 - 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].*

 - 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 `dry_depo` [1]
- Coefficients for dry deposition (v).*

- double `wet_depo` [8]
Coefficients for wet deposition (Ai, Bi, Hi, Ci, Ab, Bb, Hb, Cb).
- 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).
- 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/m²].
- int `csi_nz`
Number of altitudes of gridded CSI data.
- double `csi_z0`
Lower altitude of gridded CSI data [km].
- double `csi_z1`
Upper altitude of gridded CSI data [km].
- int `csi_nx`
Number of longitudes of gridded CSI data.
- double `csi_lon0`
Lower longitude of gridded CSI data [deg].
- double `csi_lon1`
Upper longitude of gridded CSI data [deg].
- int `csi_ny`
Number of latitudes of gridded CSI data.
- double `csi_lat0`
Lower latitude of gridded CSI data [deg].
- double `csi_lat1`
Upper latitude of gridded CSI data [deg].
- char `grid_basename` [LEN]
Basename of grid data files.
- char `grid_gpfile` [LEN]
Gnuplot file for gridded data.
- double `grid_dt_out`

- Time step for gridded data output [s].*

 - int `grid_sparse`
Sparse output in grid data files (0=no, 1=yes).
- int `grid_nz`
Number of altitudes of gridded data.
- double `grid_z0`
Lower altitude of gridded data [km].
- double `grid_z1`
Upper altitude of gridded data [km].
- int `grid_nx`
Number of longitudes of gridded data.
- double `grid_lon0`
Lower longitude of gridded data [deg].
- double `grid_lon1`
Upper longitude of gridded data [deg].
- int `grid_ny`
Number of latitudes of gridded data.
- double `grid_lat0`
Lower latitude of gridded data [deg].
- double `grid_lat1`
Upper latitude of gridded data [deg].
- char `prof_basename` [LEN]
Basename for profile output file.
- char `prof_obsfile` [LEN]
Observation data file for profile output.
- int `prof_nz`
Number of altitudes of gridded profile data.
- double `prof_z0`
Lower altitude of gridded profile data [km].
- double `prof_z1`
Upper altitude of gridded profile data [km].
- int `prof_nx`
Number of longitudes of gridded profile data.
- double `prof_lon0`
Lower longitude of gridded profile data [deg].
- double `prof_lon1`
Upper longitude of gridded profile data [deg].
- int `prof_ny`
Number of latitudes of gridded profile data.
- double `prof_lat0`
Lower latitude of gridded profile data [deg].
- double `prof_lat1`
Upper latitude of gridded profile data [deg].
- char `ens_basename` [LEN]
Basename of ensemble data file.
- char `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 width 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].

4.3.1 Detailed Description

Control parameters.

Definition at line 553 of file `libtrac.h`.

4.3.2 Field Documentation

4.3.2.1 `chunkszhint` `size_t` `ctl_t::chunkszhint`

Chunk size hint for `nc__open`.

Definition at line 556 of file `libtrac.h`.

4.3.2.2 `read_mode` `int` `ctl_t::read_mode`

Read mode for `nc__open`.

Definition at line 559 of file `libtrac.h`.

4.3.2.3 `nq` `int` `ctl_t::nq`

Number of quantities.

Definition at line 562 of file `libtrac.h`.

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

Quantity names.

Definition at line 565 of file [libtrac.h](#).

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

Quantity units.

Definition at line 568 of file [libtrac.h](#).

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

Quantity output format.

Definition at line 571 of file [libtrac.h](#).

4.3.2.7 qnt_ens `int ctl_t::qnt_ens`

Quantity array index for ensemble IDs.

Definition at line 574 of file [libtrac.h](#).

4.3.2.8 qnt_stat `int ctl_t::qnt_stat`

Quantity array index for station flag.

Definition at line 577 of file [libtrac.h](#).

4.3.2.9 qnt_m `int ctl_t::qnt_m`

Quantity array index for mass.

Definition at line 580 of file [libtrac.h](#).

4.3.2.10 `qnt_vmr` `int` `ctl_t::qnt_vmr`

Quantity array index for volume mixing ratio.

Definition at line [583](#) of file [libtrac.h](#).

4.3.2.11 `qnt_rho` `int` `ctl_t::qnt_rho`

Quantity array index for particle density.

Definition at line [586](#) of file [libtrac.h](#).

4.3.2.12 `qnt_r` `int` `ctl_t::qnt_r`

Quantity array index for particle radius.

Definition at line [589](#) of file [libtrac.h](#).

4.3.2.13 `qnt_ps` `int` `ctl_t::qnt_ps`

Quantity array index for surface pressure.

Definition at line [592](#) of file [libtrac.h](#).

4.3.2.14 `qnt_ts` `int` `ctl_t::qnt_ts`

Quantity array index for surface temperature.

Definition at line [595](#) of file [libtrac.h](#).

4.3.2.15 `qnt_zs` `int` `ctl_t::qnt_zs`

Quantity array index for surface geopotential height.

Definition at line [598](#) of file [libtrac.h](#).

4.3.2.16 qnt_us `int ctl_t::qnt_us`

Quantity array index for surface zonal wind.

Definition at line 601 of file [libtrac.h](#).

4.3.2.17 qnt_vs `int ctl_t::qnt_vs`

Quantity array index for surface meridional wind.

Definition at line 604 of file [libtrac.h](#).

4.3.2.18 qnt_pbl `int ctl_t::qnt_pbl`

Quantity array index for boundary layer pressure.

Definition at line 607 of file [libtrac.h](#).

4.3.2.19 qnt_pt `int ctl_t::qnt_pt`

Quantity array index for tropopause pressure.

Definition at line 610 of file [libtrac.h](#).

4.3.2.20 qnt_tt `int ctl_t::qnt_tt`

Quantity array index for tropopause temperature.

Definition at line 613 of file [libtrac.h](#).

4.3.2.21 qnt_zt `int ctl_t::qnt_zt`

Quantity array index for tropopause geopotential height.

Definition at line 616 of file [libtrac.h](#).

4.3.2.22 `qnt_h2ot` `int` `ctl_t::qnt_h2ot`

Quantity array index for tropopause water vapor vmr.

Definition at line 619 of file [libtrac.h](#).

4.3.2.23 `qnt_z` `int` `ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line 622 of file [libtrac.h](#).

4.3.2.24 `qnt_p` `int` `ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line 625 of file [libtrac.h](#).

4.3.2.25 `qnt_t` `int` `ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line 628 of file [libtrac.h](#).

4.3.2.26 `qnt_u` `int` `ctl_t::qnt_u`

Quantity array index for zonal wind.

Definition at line 631 of file [libtrac.h](#).

4.3.2.27 `qnt_v` `int` `ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line 634 of file [libtrac.h](#).

4.3.2.28 qnt_w `int ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line [637](#) of file [libtrac.h](#).

4.3.2.29 qnt_h2o `int ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line [640](#) of file [libtrac.h](#).

4.3.2.30 qnt_o3 `int ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line [643](#) of file [libtrac.h](#).

4.3.2.31 qnt_lwc `int ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line [646](#) of file [libtrac.h](#).

4.3.2.32 qnt_iwc `int ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line [649](#) of file [libtrac.h](#).

4.3.2.33 qnt_pct `int ctl_t::qnt_pct`

Quantity array index for cloud top pressure.

Definition at line [652](#) of file [libtrac.h](#).

4.3.2.34 `qnt_pcb` `int` `ctl_t::qnt_pcb`

Quantity array index for cloud bottom pressure.

Definition at line 655 of file [libtrac.h](#).

4.3.2.35 `qnt_cl` `int` `ctl_t::qnt_cl`

Quantity array index for total column cloud water.

Definition at line 658 of file [libtrac.h](#).

4.3.2.36 `qnt_plcl` `int` `ctl_t::qnt_plcl`

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

Definition at line 661 of file [libtrac.h](#).

4.3.2.37 `qnt_plfc` `int` `ctl_t::qnt_plfc`

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

Definition at line 664 of file [libtrac.h](#).

4.3.2.38 `qnt_pel` `int` `ctl_t::qnt_pel`

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

Definition at line 667 of file [libtrac.h](#).

4.3.2.39 `qnt_cape` `int` `ctl_t::qnt_cape`

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

Definition at line 670 of file [libtrac.h](#).

4.3.2.40 qnt_cin `int ctl_t::qnt_cin`

Quantity array index for convective inhibition (CIN).

Definition at line [673](#) of file [libtrac.h](#).

4.3.2.41 qnt_hno3 `int ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line [676](#) of file [libtrac.h](#).

4.3.2.42 qnt_oh `int ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line [679](#) of file [libtrac.h](#).

4.3.2.43 qnt_psat `int ctl_t::qnt_psat`

Quantity array index for saturation pressure over water.

Definition at line [682](#) of file [libtrac.h](#).

4.3.2.44 qnt_psice `int ctl_t::qnt_psice`

Quantity array index for saturation pressure over ice.

Definition at line [685](#) of file [libtrac.h](#).

4.3.2.45 qnt_pw `int ctl_t::qnt_pw`

Quantity array index for partial water vapor pressure.

Definition at line [688](#) of file [libtrac.h](#).

4.3.2.46 `qnt_sh` `int` `ctl_t::qnt_sh`

Quantity array index for specific humidity.

Definition at line 691 of file [libtrac.h](#).

4.3.2.47 `qnt_rh` `int` `ctl_t::qnt_rh`

Quantity array index for relative humidity over water.

Definition at line 694 of file [libtrac.h](#).

4.3.2.48 `qnt_rhice` `int` `ctl_t::qnt_rhice`

Quantity array index for relative humidity over ice.

Definition at line 697 of file [libtrac.h](#).

4.3.2.49 `qnt_theta` `int` `ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line 700 of file [libtrac.h](#).

4.3.2.50 `qnt_zeta` `int` `ctl_t::qnt_zeta`

Quantity array index for zeta vertical coordinate.

Definition at line 703 of file [libtrac.h](#).

4.3.2.51 `qnt_tvirt` `int` `ctl_t::qnt_tvirt`

Quantity array index for virtual temperature.

Definition at line 706 of file [libtrac.h](#).

4.3.2.52 qnt_lapse `int ctl_t::qnt_lapse`

Quantity array index for lapse rate.

Definition at line [709](#) of file [libtrac.h](#).

4.3.2.53 qnt_vh `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line [712](#) of file [libtrac.h](#).

4.3.2.54 qnt_vz `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line [715](#) of file [libtrac.h](#).

4.3.2.55 qnt_pv `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line [718](#) of file [libtrac.h](#).

4.3.2.56 qnt_tdew `int ctl_t::qnt_tdew`

Quantity array index for dew point temperature.

Definition at line [721](#) of file [libtrac.h](#).

4.3.2.57 qnt_tice `int ctl_t::qnt_tice`

Quantity array index for T_ice.

Definition at line [724](#) of file [libtrac.h](#).

4.3.2.58 `qnt_tsts` `int` `ctl_t::qnt_tsts`

Quantity array index for T_STS.

Definition at line 727 of file [libtrac.h](#).

4.3.2.59 `qnt_tnat` `int` `ctl_t::qnt_tnat`

Quantity array index for T_NAT.

Definition at line 730 of file [libtrac.h](#).

4.3.2.60 `direction` `int` `ctl_t::direction`

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

Definition at line 733 of file [libtrac.h](#).

4.3.2.61 `t_start` `double` `ctl_t::t_start`

Start time of simulation [s].

Definition at line 736 of file [libtrac.h](#).

4.3.2.62 `t_stop` `double` `ctl_t::t_stop`

Stop time of simulation [s].

Definition at line 739 of file [libtrac.h](#).

4.3.2.63 `dt_mod` `double` `ctl_t::dt_mod`

Time step of simulation [s].

Definition at line 742 of file [libtrac.h](#).

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

Basename for meteorological data.

Definition at line 745 of file [libtrac.h](#).

4.3.2.65 dt_met `double ctl_t::dt_met`

Time step of meteorological data [s].

Definition at line 748 of file [libtrac.h](#).

4.3.2.66 met_dx `int ctl_t::met_dx`

Stride for longitudes.

Definition at line 751 of file [libtrac.h](#).

4.3.2.67 met_dy `int ctl_t::met_dy`

Stride for latitudes.

Definition at line 754 of file [libtrac.h](#).

4.3.2.68 met_dp `int ctl_t::met_dp`

Stride for pressure levels.

Definition at line 757 of file [libtrac.h](#).

4.3.2.69 met_sx `int ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 760 of file [libtrac.h](#).

4.3.2.70 `met_sy` `int` `ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 763 of file [libtrac.h](#).

4.3.2.71 `met_sp` `int` `ctl_t::met_sp`

Smoothing for pressure levels.

Definition at line 766 of file [libtrac.h](#).

4.3.2.72 `met_detrend` `double` `ctl_t::met_detrend`

FWHM of horizontal Gaussian used for detrending [km].

Definition at line 769 of file [libtrac.h](#).

4.3.2.73 `met_np` `int` `ctl_t::met_np`

Number of target pressure levels.

Definition at line 772 of file [libtrac.h](#).

4.3.2.74 `met_p` `double` `ctl_t::met_p` [\[EP\]](#)

Target pressure levels [hPa].

Definition at line 775 of file [libtrac.h](#).

4.3.2.75 `met_geopot_sx` `int` `ctl_t::met_geopot_sx`

Longitudinal smoothing of geopotential heights.

Definition at line 778 of file [libtrac.h](#).

4.3.2.76 met_geopot_sy `int ctl_t::met_geopot_sy`

Latitudinal smoothing of geopotential heights.

Definition at line 781 of file [libtrac.h](#).

4.3.2.77 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 785 of file [libtrac.h](#).

4.3.2.78 met_tropo_lapse `double ctl_t::met_tropo_lapse`

WMO tropopause lapse rate [K/km].

Definition at line 788 of file [libtrac.h](#).

4.3.2.79 met_tropo_nlev `int ctl_t::met_tropo_nlev`

WMO tropopause layer width (number of levels).

Definition at line 791 of file [libtrac.h](#).

4.3.2.80 met_tropo_lapse_sep `double ctl_t::met_tropo_lapse_sep`

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

Definition at line 794 of file [libtrac.h](#).

4.3.2.81 met_tropo_nlev_sep `int ctl_t::met_tropo_nlev_sep`

WMO tropopause separation layer width (number of levels).

Definition at line 797 of file [libtrac.h](#).

4.3.2.82 `met_tropo_pv` `double ctl_t::met_tropo_pv`

Dynamical tropopause potential vorticity threshold [PVU].

Definition at line 800 of file [libtrac.h](#).

4.3.2.83 `met_tropo_theta` `double ctl_t::met_tropo_theta`

Dynamical tropopause potential temperature threshold [K].

Definition at line 803 of file [libtrac.h](#).

4.3.2.84 `met_tropo_spline` `int ctl_t::met_tropo_spline`

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

Definition at line 806 of file [libtrac.h](#).

4.3.2.85 `met_cloud` `double ctl_t::met_cloud`

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

Definition at line 809 of file [libtrac.h](#).

4.3.2.86 `met_dt_out` `double ctl_t::met_dt_out`

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

Definition at line 812 of file [libtrac.h](#).

4.3.2.87 `met_cache` `int ctl_t::met_cache`

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

Definition at line 815 of file [libtrac.h](#).

4.3.2.88 isosurf `int ctl_t::isosurf`

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

Definition at line 819 of file [libtrac.h](#).

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

Balloon position filename.

Definition at line 822 of file [libtrac.h](#).

4.3.2.90 advect `int ctl_t::advect`

Advection scheme (0=midpoint, 1=Runge-Kutta).

Definition at line 825 of file [libtrac.h](#).

4.3.2.91 reflect `int ctl_t::reflect`

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

Definition at line 828 of file [libtrac.h](#).

4.3.2.92 turb_dx_trop `double ctl_t::turb_dx_trop`

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

Definition at line 831 of file [libtrac.h](#).

4.3.2.93 turb_dx_strat `double ctl_t::turb_dx_strat`

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

Definition at line 834 of file [libtrac.h](#).

4.3.2.94 `turb_dz_trop` `double ctl_t::turb_dz_trop`

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

Definition at line 837 of file [libtrac.h](#).

4.3.2.95 `turb_dz_strat` `double ctl_t::turb_dz_strat`

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

Definition at line 840 of file [libtrac.h](#).

4.3.2.96 `turb_mesox` `double ctl_t::turb_mesox`

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 843 of file [libtrac.h](#).

4.3.2.97 `turb_mesoz` `double ctl_t::turb_mesoz`

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 846 of file [libtrac.h](#).

4.3.2.98 `conv_cape` `double ctl_t::conv_cape`

CAPE threshold for convection module [J/kg].

Definition at line 849 of file [libtrac.h](#).

4.3.2.99 `conv_cin` `double ctl_t::conv_cin`

CIN threshold for convection module [J/kg].

Definition at line 852 of file [libtrac.h](#).

4.3.2.100 conv_wmax `double ctl_t::conv_wmax`

Maximum vertical velocity for convection module [m/s].

Definition at line [855](#) of file [libtrac.h](#).

4.3.2.101 conv_wcape `double ctl_t::conv_wcape`

Limit vertical velocity based on CAPE (0=no, 1=yes).

Definition at line [858](#) of file [libtrac.h](#).

4.3.2.102 conv_dt `double ctl_t::conv_dt`

Time interval for convection module [s].

Definition at line [861](#) of file [libtrac.h](#).

4.3.2.103 conv_mix_bot `int ctl_t::conv_mix_bot`

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

Definition at line [864](#) of file [libtrac.h](#).

4.3.2.104 conv_mix_top `int ctl_t::conv_mix_top`

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

Definition at line [867](#) of file [libtrac.h](#).

4.3.2.105 bound_mass `double ctl_t::bound_mass`

Boundary conditions mass per particle [kg].

Definition at line [870](#) of file [libtrac.h](#).

4.3.2.106 `bound_vmr` `double ctl_t::bound_vmr`

Boundary conditions volume mixing ratio [ppv].

Definition at line 873 of file [libtrac.h](#).

4.3.2.107 `bound_lat0` `double ctl_t::bound_lat0`

Boundary conditions minimum longitude [deg].

Definition at line 876 of file [libtrac.h](#).

4.3.2.108 `bound_lat1` `double ctl_t::bound_lat1`

Boundary conditions maximum longitude [deg].

Definition at line 879 of file [libtrac.h](#).

4.3.2.109 `bound_p0` `double ctl_t::bound_p0`

Boundary conditions bottom pressure [hPa].

Definition at line 882 of file [libtrac.h](#).

4.3.2.110 `bound_p1` `double ctl_t::bound_p1`

Boundary conditions top pressure [hPa].

Definition at line 885 of file [libtrac.h](#).

4.3.2.111 `bound_dps` `double ctl_t::bound_dps`

Boundary conditions delta to surface pressure [hPa].

Definition at line 888 of file [libtrac.h](#).

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

Species.

Definition at line 891 of file [libtrac.h](#).

4.3.2.113 molmass `double ctl_t::molmass`

Molar mass [g/mol].

Definition at line 894 of file [libtrac.h](#).

4.3.2.114 tdec_trop `double ctl_t::tdec_trop`

Life time of particles (troposphere) [s].

Definition at line 897 of file [libtrac.h](#).

4.3.2.115 tdec_strat `double ctl_t::tdec_strat`

Life time of particles (stratosphere) [s].

Definition at line 900 of file [libtrac.h](#).

4.3.2.116 oh_chem_reaction `int ctl_t::oh_chem_reaction`

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

Definition at line 903 of file [libtrac.h](#).

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

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

Definition at line 906 of file [libtrac.h](#).

4.3.2.118 `dry_depo` `double ctl_t::dry_depo[1]`

Coefficients for dry deposition (v).

Definition at line 909 of file [libtrac.h](#).

4.3.2.119 `wet_depo` `double ctl_t::wet_depo[8]`

Coefficients for wet deposition (Ai, Bi, Hi, Ci, Ab, Bb, Hb, Cb).

Definition at line 912 of file [libtrac.h](#).

4.3.2.120 `psc_h2o` `double ctl_t::psc_h2o`

H2O volume mixing ratio for PSC analysis.

Definition at line 915 of file [libtrac.h](#).

4.3.2.121 `psc_hno3` `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

Definition at line 918 of file [libtrac.h](#).

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

Baseline of atmospheric data files.

Definition at line 921 of file [libtrac.h](#).

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

Gnuplot file for atmospheric data.

Definition at line 924 of file [libtrac.h](#).

4.3.2.124 atm_dt_out `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

Definition at line 927 of file [libtrac.h](#).

4.3.2.125 atm_filter `int ctl_t::atm_filter`

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

Definition at line 930 of file [libtrac.h](#).

4.3.2.126 atm_stride `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line 933 of file [libtrac.h](#).

4.3.2.127 atm_type `int ctl_t::atm_type`

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

Definition at line 936 of file [libtrac.h](#).

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

Basename of CSI data files.

Definition at line 939 of file [libtrac.h](#).

4.3.2.129 csi_dt_out `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line 942 of file [libtrac.h](#).

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

Observation data file for CSI analysis.

Definition at line 945 of file [libtrac.h](#).

4.3.2.131 `csi_obsmin` `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line 948 of file [libtrac.h](#).

4.3.2.132 `csi_modmin` `double ctl_t::csi_modmin`

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

Definition at line 951 of file [libtrac.h](#).

4.3.2.133 `csi_nz` `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line 954 of file [libtrac.h](#).

4.3.2.134 `csi_z0` `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

Definition at line 957 of file [libtrac.h](#).

4.3.2.135 `csi_z1` `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

Definition at line 960 of file [libtrac.h](#).

4.3.2.136 `csi_nx` `int` `ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

Definition at line 963 of file [libtrac.h](#).

4.3.2.137 `csi_lon0` `double` `ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 966 of file [libtrac.h](#).

4.3.2.138 `csi_lon1` `double` `ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 969 of file [libtrac.h](#).

4.3.2.139 `csi_ny` `int` `ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 972 of file [libtrac.h](#).

4.3.2.140 `csi_lat0` `double` `ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 975 of file [libtrac.h](#).

4.3.2.141 `csi_lat1` `double` `ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line 978 of file [libtrac.h](#).

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

Basename of grid data files.

Definition at line 981 of file [libtrac.h](#).

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

Gnuplot file for gridded data.

Definition at line 984 of file [libtrac.h](#).

4.3.2.144 `grid_dt_out` `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

Definition at line 987 of file [libtrac.h](#).

4.3.2.145 `grid_sparse` `int ctl_t::grid_sparse`

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

Definition at line 990 of file [libtrac.h](#).

4.3.2.146 `grid_nz` `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line 993 of file [libtrac.h](#).

4.3.2.147 `grid_z0` `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line 996 of file [libtrac.h](#).

4.3.2.148 **grid_z1** `double ctl_t::grid_z1`

Upper altitude of gridded data [km].

Definition at line 999 of file [libtrac.h](#).

4.3.2.149 **grid_nx** `int ctl_t::grid_nx`

Number of longitudes of gridded data.

Definition at line 1002 of file [libtrac.h](#).

4.3.2.150 **grid_lon0** `double ctl_t::grid_lon0`

Lower longitude of gridded data [deg].

Definition at line 1005 of file [libtrac.h](#).

4.3.2.151 **grid_lon1** `double ctl_t::grid_lon1`

Upper longitude of gridded data [deg].

Definition at line 1008 of file [libtrac.h](#).

4.3.2.152 **grid_ny** `int ctl_t::grid_ny`

Number of latitudes of gridded data.

Definition at line 1011 of file [libtrac.h](#).

4.3.2.153 **grid_lat0** `double ctl_t::grid_lat0`

Lower latitude of gridded data [deg].

Definition at line 1014 of file [libtrac.h](#).

4.3.2.154 `grid_lat1` `double ctl_t::grid_lat1`

Upper latitude of gridded data [deg].

Definition at line [1017](#) of file [libtrac.h](#).

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

Basename for profile output file.

Definition at line [1020](#) of file [libtrac.h](#).

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

Observation data file for profile output.

Definition at line [1023](#) of file [libtrac.h](#).

4.3.2.157 `prof_nz` `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line [1026](#) of file [libtrac.h](#).

4.3.2.158 `prof_z0` `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line [1029](#) of file [libtrac.h](#).

4.3.2.159 `prof_z1` `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line [1032](#) of file [libtrac.h](#).

4.3.2.160 `prof_nx` `int` `ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line [1035](#) of file [libtrac.h](#).

4.3.2.161 `prof_lon0` `double` `ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line [1038](#) of file [libtrac.h](#).

4.3.2.162 `prof_lon1` `double` `ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line [1041](#) of file [libtrac.h](#).

4.3.2.163 `prof_ny` `int` `ctl_t::prof_ny`

Number of latitudes of gridded profile data.

Definition at line [1044](#) of file [libtrac.h](#).

4.3.2.164 `prof_lat0` `double` `ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

Definition at line [1047](#) of file [libtrac.h](#).

4.3.2.165 `prof_lat1` `double` `ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

Definition at line [1050](#) of file [libtrac.h](#).

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

Basename of ensemble data file.

Definition at line 1053 of file [libtrac.h](#).

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

Basename of sample data file.

Definition at line 1056 of file [libtrac.h](#).

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

Observation data file for sample output.

Definition at line 1059 of file [libtrac.h](#).

4.3.2.169 sample_dx `double ctl_t::sample_dx`

Horizontal radius for sample output [km].

Definition at line 1062 of file [libtrac.h](#).

4.3.2.170 sample_dz `double ctl_t::sample_dz`

Layer width for sample output [km].

Definition at line 1065 of file [libtrac.h](#).

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

Basename of station data file.

Definition at line 1068 of file [libtrac.h](#).

4.3.2.172 `stat_lon` `double ctl_t::stat_lon`

Longitude of station [deg].

Definition at line [1071](#) of file [libtrac.h](#).

4.3.2.173 `stat_lat` `double ctl_t::stat_lat`

Latitude of station [deg].

Definition at line [1074](#) of file [libtrac.h](#).

4.3.2.174 `stat_r` `double ctl_t::stat_r`

Search radius around station [km].

Definition at line [1077](#) of file [libtrac.h](#).

4.3.2.175 `stat_t0` `double ctl_t::stat_t0`

Start time for station output [s].

Definition at line [1080](#) of file [libtrac.h](#).

4.3.2.176 `stat_t1` `double ctl_t::stat_t1`

Stop time for station output [s].

Definition at line [1083](#) of file [libtrac.h](#).

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

- [libtrac.h](#)

4.4 `met_t` Struct Reference

Meteorological data.

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

Data Fields

- double [time](#)
Time [s].
- int [nx](#)
Number of longitudes.
- int [ny](#)
Number of latitudes.
- int [np](#)
Number of pressure levels.
- double [lon](#) [[EX](#)]
Longitude [deg].
- double [lat](#) [[EY](#)]
Latitude [deg].
- double [p](#) [[EP](#)]
Pressure [hPa].
- float [ps](#) [[EX](#)][[EY](#)]
Surface pressure [hPa].
- float [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](#)]

- float **z** [EX][EY][EP]
Convective inhibition [J/kg].
- float **t** [EX][EY][EP]
Geopotential height at model levels [km].
- float **u** [EX][EY][EP]
Temperature [K].
- float **v** [EX][EY][EP]
Zonal wind [m/s].
- float **w** [EX][EY][EP]
Meridional wind [m/s].
- float **pv** [EX][EY][EP]
Vertical velocity [hPa/s].
- float **h2o** [EX][EY][EP]
Potential vorticity [PVU].
- float **o3** [EX][EY][EP]
Water vapor volume mixing ratio [1].
- float **lwc** [EX][EY][EP]
Ozone volume mixing ratio [1].
- float **iw** [EX][EY][EP]
Cloud liquid water content [kg/kg].
- float **pl** [EX][EY][EP]
Cloud ice water content [kg/kg].
- float **pl** [EX][EY][EP]
Pressure on model levels [hPa].

4.4.1 Detailed Description

Meteorological data.

Definition at line 1137 of file `libtrac.h`.

4.4.2 Field Documentation

4.4.2.1 `time` double met_t::time

Time [s].

Definition at line 1140 of file `libtrac.h`.

4.4.2.2 `nx` int met_t::nx

Number of longitudes.

Definition at line 1143 of file `libtrac.h`.

4.4.2.3 ny `int met_t::ny`

Number of latitudes.

Definition at line 1146 of file [libtrac.h](#).

4.4.2.4 np `int met_t::np`

Number of pressure levels.

Definition at line 1149 of file [libtrac.h](#).

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

Longitude [deg].

Definition at line 1152 of file [libtrac.h](#).

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

Latitude [deg].

Definition at line 1155 of file [libtrac.h](#).

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

Pressure [hPa].

Definition at line 1158 of file [libtrac.h](#).

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

Surface pressure [hPa].

Definition at line 1161 of file [libtrac.h](#).

4.4.2.9 ts `float met_t::ts`[\[EX\]](#) [\[EY\]](#)

Surface temperature [K].

Definition at line [1164](#) of file [libtrac.h](#).

4.4.2.10 zs `float met_t::zs`[\[EX\]](#) [\[EY\]](#)

Surface geopotential height [km].

Definition at line [1167](#) of file [libtrac.h](#).

4.4.2.11 us `float met_t::us`[\[EX\]](#) [\[EY\]](#)

Surface zonal wind [m/s].

Definition at line [1170](#) of file [libtrac.h](#).

4.4.2.12 vs `float met_t::vs`[\[EX\]](#) [\[EY\]](#)

Surface meridional wind [m/s].

Definition at line [1173](#) of file [libtrac.h](#).

4.4.2.13 pbl `float met_t::pbl`[\[EX\]](#) [\[EY\]](#)

Boundary layer pressure [hPa].

Definition at line [1176](#) of file [libtrac.h](#).

4.4.2.14 pt `float met_t::pt`[\[EX\]](#) [\[EY\]](#)

Tropopause pressure [hPa].

Definition at line [1179](#) of file [libtrac.h](#).

4.4.2.15 tt float met_t::tt [\[EX\]](#) [\[EY\]](#)

Tropopause temperature [K].

Definition at line [1182](#) of file [libtrac.h](#).

4.4.2.16 zt float met_t::zt [\[EX\]](#) [\[EY\]](#)

Tropopause geopotential height [km].

Definition at line [1185](#) of file [libtrac.h](#).

4.4.2.17 h2ot float met_t::h2ot [\[EX\]](#) [\[EY\]](#)

Tropopause water vapor vmr [ppv].

Definition at line [1188](#) of file [libtrac.h](#).

4.4.2.18 pct float met_t::pct [\[EX\]](#) [\[EY\]](#)

Cloud top pressure [hPa].

Definition at line [1191](#) of file [libtrac.h](#).

4.4.2.19 pcb float met_t::pcb [\[EX\]](#) [\[EY\]](#)

Cloud bottom pressure [hPa].

Definition at line [1194](#) of file [libtrac.h](#).

4.4.2.20 cl float met_t::cl [\[EX\]](#) [\[EY\]](#)

Total column cloud water [kg/m²].

Definition at line [1197](#) of file [libtrac.h](#).

4.4.2.21 plcl float met_t::plcl[EX][EY]

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

Definition at line 1200 of file libtrac.h.

4.4.2.22 plfc float met_t::plfc[EX][EY]

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

Definition at line 1203 of file libtrac.h.

4.4.2.23 pel float met_t::pel[EX][EY]

Pressure at equilibrium level [hPa].

Definition at line 1206 of file libtrac.h.

4.4.2.24 cape float met_t::cape[EX][EY]

Convective available potential energy [J/kg].

Definition at line 1209 of file libtrac.h.

4.4.2.25 cin float met_t::cin[EX][EY]

Convective inhibition [J/kg].

Definition at line 1212 of file libtrac.h.

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

Geopotential height at model levels [km].

Definition at line 1215 of file libtrac.h.

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

Temperature [K].

Definition at line 1218 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 1221 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 1224 of file libtrac.h.

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

Vertical velocity [hPa/s].

Definition at line 1227 of file libtrac.h.

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

Potential vorticity [PVU].

Definition at line 1230 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 1233 of file libtrac.h.

4.4.2.33 o3 `float met_t::o3`[\[EX\]](#) [\[EY\]](#) [\[EP\]](#)

Ozone volume mixing ratio [1].

Definition at line [1236](#) of file [libtrac.h](#).

4.4.2.34 lwc `float met_t::lwc`[\[EX\]](#) [\[EY\]](#) [\[EP\]](#)

Cloud liquid water content [kg/kg].

Definition at line [1239](#) of file [libtrac.h](#).

4.4.2.35 iwc `float met_t::iwc`[\[EX\]](#) [\[EY\]](#) [\[EP\]](#)

Cloud ice water content [kg/kg].

Definition at line [1242](#) of file [libtrac.h](#).

4.4.2.36 pl `float met_t::pl`[\[EX\]](#) [\[EY\]](#) [\[EP\]](#)

Pressure on model levels [hPa].

Definition at line [1245](#) 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

```
#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[])`

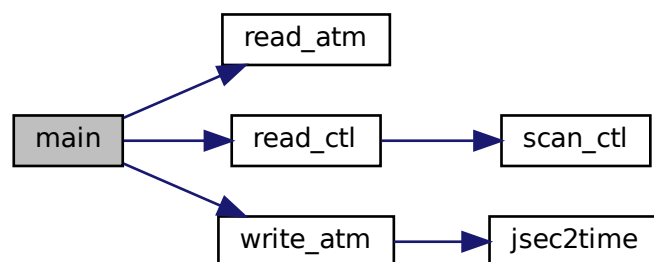
Definition at line 27 of file [atm_conv.c](#).

```

00029     {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm;
00034
00035     /* Check arguments... */
00036     if (argc < 6)
00037         ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038              " <atm_out> <atm_out_type>");
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042
00043     /* Read control parameters... */
00044     read_ctl(argv[1], argc, argv, &ctl);
00045
00046     /* Read atmospheric data... */
00047     ctl.atm_type = atoi(argv[3]);
00048     if (!read_atm(argv[2], &ctl, atm))
00049         ERRMSG("Cannot open file!");
00050
00051     /* Write atmospheric data... */
00052     ctl.atm_type = atoi(argv[5]);
00053     write_atm(argv[4], &ctl, atm, 0);
00054
00055     /* Free... */
00056     free(atm);
00057
00058     return EXIT_SUCCESS;
00059 }

```

Here is the call graph for this function:



5.2 atm_conv.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.

```

```

00008
00009 MPTRAC is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU General Public License for more details.
00013
00014 You should have received a copy of the GNU General Public License
00015 along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm;
00034
00035     /* Check arguments... */
00036     if (argc < 6)
00037         ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038             " <atm_out> <atm_out_type>");
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042
00043     /* Read control parameters... */
00044     read_ctl(argv[1], argc, argv, &ctl);
00045
00046     /* Read atmospheric data... */
00047     ctl.atm_type = atoi(argv[3]);
00048     if (!read_atm(argv[2], &ctl, atm))
00049         ERRMSG("Cannot open file!");
00050
00051     /* Write atmospheric data... */
00052     ctl.atm_type = atoi(argv[5]);
00053     write_atm(argv[4], &ctl, atm, 0);
00054
00055     /* Free... */
00056     free(atm);
00057
00058     return EXIT_SUCCESS;
00059 }

```

5.3 atm_dist.c File Reference

```
#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](#).

```

00029     {
00030
00031         ctl_t ctl;
00032
00033         atm_t *atm1, *atm2;
00034
00035         FILE *out;
00036
00037         char tstr[LEN];
00038
00039         double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
00040             *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041             *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00042             t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work, zscore;
00043
00044         int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00045
00046         /* Allocate... */
00047         ALLOC(atm1, atm_t, 1);
00048         ALLOC(atm2, atm_t, 1);
00049         ALLOC(lon1_old, double,
00050             NP);
00051         ALLOC(lat1_old, double,
00052             NP);
00053         ALLOC(z1_old, double,
00054             NP);
00055         ALLOC(lh1, double,
00056             NP);
00057         ALLOC(lv1, double,
00058             NP);
00059         ALLOC(lon2_old, double,
00060             NP);
00061         ALLOC(lat2_old, double,
00062             NP);
00063         ALLOC(z2_old, double,
00064             NP);
00065         ALLOC(lh2, double,
00066             NP);
00067         ALLOC(lv2, double,
00068             NP);
00069         ALLOC(ahtd, double,
00070             NP);
00071         ALLOC(avtd, double,
00072             NP);
00073         ALLOC(aqtd, double,
00074             NP * NQ);
00075         ALLOC(rhtd, double,
00076             NP);
00077         ALLOC(rvtd, double,
00078             NP);
00079         ALLOC(rqtd, double,
00080             NP * NQ);
00081         ALLOC(work, double,
00082             NP);
00083
00084         /* Check arguments... */
00085         if (argc < 6)
00086             ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087                 " [<atm2a> <atm2b> ...]");
00088
00089         /* Read control parameters... */
00090         read_ctl(argv[1], argc, argv, &ctl);
00091         ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092         p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093         p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094         lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095         lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096         lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097         lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098         zscore = scan_ctl(argv[1], argc, argv, "DIST_ZSCORE", -1, "-999", NULL);
00099
00100         /* Write info... */
00101         LOG(1, "Write transport deviations: %s", argv[2]);
00102
00103         /* Create output file... */
00104         if (!(out = fopen(argv[2], "w")))
00105             ERRMSG("Cannot create file!");
00106
00107         /* Write header... */
00108         fprintf(out,

```

```

00109         "# $1 = time [s]\n"
00110         "# $2 = time difference [s]\n"
00111         "# $3 = absolute horizontal distance (%s) [km]\n"
00112         "# $4 = relative horizontal distance (%s) [%%]\n"
00113         "# $5 = absolute vertical distance (%s) [km]\n"
00114         "# $6 = relative vertical distance (%s) [%%]\n",
00115         argv[3], argv[3], argv[3], argv[3]);
00116     for (iq = 0; iq < ctl.nq; iq++)
00117         fprintf(out,
00118             "# %d = %s absolute difference (%s) [%s]\n"
00119             "# %d = %s relative difference (%s) [%%]\n",
00120             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00121             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00122     fprintf(out, "# %d = number of particles\n\n", 7 + 2 * ctl.nq);
00123
00124     /* Loop over file pairs... */
00125     for (f = 4; f < argc; f += 2) {
00126
00127         /* Read atmospheric data... */
00128         if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00129             continue;
00130
00131         /* Check if structs match... */
00132         if (atm1->np != atm2->np)
00133             ERRMSG("Different numbers of particles!");
00134
00135         /* Get time from filename... */
00136         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00137         year = atoi(tstr);
00138         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00139         mon = atoi(tstr);
00140         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00141         day = atoi(tstr);
00142         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00143         hour = atoi(tstr);
00144         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00145         min = atoi(tstr);
00146         time2jsec(year, mon, day, hour, min, 0, 0, &t);
00147
00148         /* Check time... */
00149         if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00150             || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00151             ERRMSG("Cannot read time from filename!");
00152
00153         /* Save initial time... */
00154         if (!init) {
00155             init = 1;
00156             t0 = t;
00157         }
00158
00159         /* Init... */
00160         np = 0;
00161         for (ip = 0; ip < atm1->np; ip++) {
00162             ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00163             for (iq = 0; iq < ctl.nq; iq++)
00164                 aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00165         }
00166
00167         /* Loop over air parcels... */
00168         for (ip = 0; ip < atm1->np; ip++) {
00169
00170             /* Check data... */
00171             if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00172                 continue;
00173
00174             /* Check ensemble index... */
00175             if (ctl.qnt_ens > 0
00176                 && (atm1->q[ctl.qnt_ens][ip] != ens
00177                     || atm2->q[ctl.qnt_ens][ip] != ens))
00178                 continue;
00179
00180             /* Check spatial range... */
00181             if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00182                 || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00183                 || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00184                 continue;
00185             if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00186                 || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00187                 || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00188                 continue;
00189
00190             /* Convert coordinates... */
00191             geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00192             geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00193             z1 = Z(atm1->p[ip]);
00194             z2 = Z(atm2->p[ip]);
00195

```

```

00196      /* Calculate absolute transport deviations... */
00197      ahtd[np] = DIST(x1, x2);
00198      avtd[np] = z1 - z2;
00199      for (iq = 0; iq < ctl.nq; iq++)
00200          aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00201
00202      /* Calculate relative transport deviations... */
00203      if (f > 4) {
00204
00205          /* Get trajectory lengths... */
00206          geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00207          lh1[ip] += DIST(x0, x1);
00208          lv1[ip] += fabs(z1_old[ip] - z1);
00209
00210          geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00211          lh2[ip] += DIST(x0, x2);
00212          lv2[ip] += fabs(z2_old[ip] - z2);
00213
00214          /* Get relative transport deviations... */
00215          if (lh1[ip] + lh2[ip] > 0)
00216              rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00217          if (lv1[ip] + lv2[ip] > 0)
00218              rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00219      }
00220
00221      /* Get relative transport deviations... */
00222      for (iq = 0; iq < ctl.nq; iq++)
00223          rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00224              / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00225
00226      /* Save positions of air parcels... */
00227      lon1_old[ip] = atm1->lon[ip];
00228      lat1_old[ip] = atm1->lat[ip];
00229      z1_old[ip] = z1;
00230
00231      lon2_old[ip] = atm2->lon[ip];
00232      lat2_old[ip] = atm2->lat[ip];
00233      z2_old[ip] = z2;
00234
00235      /* Increment air parcel counter... */
00236      np++;
00237  }
00238
00239  /* Filter data... */
00240  if (zscore > 0 && np > 1) {
00241
00242      /* Get means and standard deviations of transport deviations... */
00243      size_t n = (size_t) np;
00244      double muh = gsl_stats_mean(ahtd, 1, n);
00245      double muv = gsl_stats_mean(avtd, 1, n);
00246      double sigh = gsl_stats_sd(ahtd, 1, n);
00247      double sigv = gsl_stats_sd(avtd, 1, n);
00248
00249      /* Filter data... */
00250      np = 0;
00251      for (size_t i = 0; i < n; i++)
00252          if (fabs((ahtd[i] - muh) / sigh) < zscore
00253              && fabs((avtd[i] - muv) / sigv) < zscore) {
00254              ahtd[np] = ahtd[i];
00255              rhtd[np] = rhtd[i];
00256              avtd[np] = avtd[i];
00257              rvtd[np] = rvtd[i];
00258              for (iq = 0; iq < ctl.nq; iq++) {
00259                  aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
00260                  rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
00261              }
00262              np++;
00263          }
00264      }
00265
00266      /* Get statistics... */
00267      if (strcmp(argv[3], "mean") == 0) {
00268          ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00269          rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00270          avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00271          rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00272          for (iq = 0; iq < ctl.nq; iq++) {
00273              aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00274              rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00275          }
00276      } else if (strcmp(argv[3], "stddev") == 0) {
00277          ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00278          rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00279          avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00280          rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00281          for (iq = 0; iq < ctl.nq; iq++) {
00282              aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);

```

```

00283     rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00284 }
00285 } else if (strcmp(argv[3], "min") == 0) {
00286     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00287     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00288     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00289     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00290     for (iq = 0; iq < ctl.nq; iq++) {
00291         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00292         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00293     }
00294 } else if (strcmp(argv[3], "max") == 0) {
00295     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00296     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00297     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00298     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00299     for (iq = 0; iq < ctl.nq; iq++) {
00300         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00301         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00302     }
00303 } else if (strcmp(argv[3], "skew") == 0) {
00304     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00305     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00306     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00307     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00308     for (iq = 0; iq < ctl.nq; iq++) {
00309         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00310         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00311     }
00312 } else if (strcmp(argv[3], "kurt") == 0) {
00313     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00314     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00315     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00316     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00317     for (iq = 0; iq < ctl.nq; iq++) {
00318         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00319         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00320     }
00321 } else if (strcmp(argv[3], "absdev") == 0) {
00322     ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
00323     rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
00324     avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
00325     rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
00326     for (iq = 0; iq < ctl.nq; iq++) {
00327         aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
00328         rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
00329     }
00330 } else if (strcmp(argv[3], "median") == 0) {
00331     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00332     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00333     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00334     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00335     for (iq = 0; iq < ctl.nq; iq++) {
00336         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00337         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00338     }
00339 } else if (strcmp(argv[3], "mad") == 0) {
00340     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00341     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00342     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00343     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00344     for (iq = 0; iq < ctl.nq; iq++) {
00345         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00346         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00347     }
00348 } else
00349     ERRMSG("Unknown parameter!");
00350
00351 /* Write output... */
00352 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00353         ahtdm, rhtdm, avtdm, rvtdm);
00354 for (iq = 0; iq < ctl.nq; iq++) {
00355     fprintf(out, " ");
00356     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00357     fprintf(out, " ");
00358     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00359 }
00360 fprintf(out, "\n", np);
00361 }
00362
00363 /* Close file... */
00364 fclose(out);
00365
00366 /* Free... */
00367 free(atm1);
00368 free(atm2);
00369 free(lonl_old);

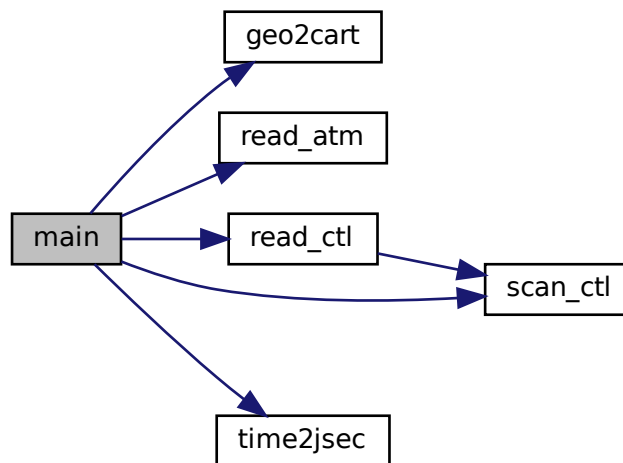
```

```

00370     free(lat1_old);
00371     free(zl_old);
00372     free(lh1);
00373     free(lv1);
00374     free(lon2_old);
00375     free(lat2_old);
00376     free(z2_old);
00377     free(lh2);
00378     free(lv2);
00379     free(ahtd);
00380     free(avtd);
00381     free(aqtd);
00382     free(rhtd);
00383     free(rvtd);
00384     free(rqtd);
00385     free(work);
00386
00387     return EXIT_SUCCESS;
00388 }

```

Here is the call graph for this function:



5.4 atm_dist.c

```

00001  /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  int main(
00028      int argc,
00029      char *argv[]) {
00030

```

```

00031     ctl_t   ctl;
00032
00033     atm_t   *atm1, *atm2;
00034
00035     FILE *out;
00036
00037     char tstr[LEN];
00038
00039     double *ahtd, *aqtd, *avtd, ahtdm, aqtdm[NQ], avtdm, lat0, lat1,
00040           *lat1_old, *lat2_old, *lh1, *lh2, lon0, lon1, *lon1_old, *lon2_old,
00041           *lv1, *lv2, p0, p1, *rhtd, *rqtd, *rvtd, rhtdm, rqtdm[NQ], rvtdm,
00042           t, t0 = 0, x0[3], x1[3], x2[3], z1, *z1_old, z2, *z2_old, *work, zscore;
00043
00044     int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00045
00046     /* Allocate... */
00047     ALLOC(atm1, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049     ALLOC(lon1_old, double,
00050           NP);
00051     ALLOC(lat1_old, double,
00052           NP);
00053     ALLOC(z1_old, double,
00054           NP);
00055     ALLOC(lh1, double,
00056           NP);
00057     ALLOC(lv1, double,
00058           NP);
00059     ALLOC(lon2_old, double,
00060           NP);
00061     ALLOC(lat2_old, double,
00062           NP);
00063     ALLOC(z2_old, double,
00064           NP);
00065     ALLOC(lh2, double,
00066           NP);
00067     ALLOC(lv2, double,
00068           NP);
00069     ALLOC(ahtd, double,
00070           NP);
00071     ALLOC(avtd, double,
00072           NP);
00073     ALLOC(aqtd, double,
00074           NP * NQ);
00075     ALLOC(rhtd, double,
00076           NP);
00077     ALLOC(rvtd, double,
00078           NP);
00079     ALLOC(rqtd, double,
00080           NP * NQ);
00081     ALLOC(work, double,
00082           NP);
00083
00084     /* Check arguments... */
00085     if (argc < 6)
00086         ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087               "  [<atm2a> <atm2b> ...]");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092     p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093     p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094     lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095     lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096     lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097     lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098     zscore = scan_ctl(argv[1], argc, argv, "DIST_ZSCORE", -1, "-999", NULL);
00099
00100     /* Write info... */
00101     LOG(1, "Write transport deviations: %s", argv[2]);
00102
00103     /* Create output file... */
00104     if (!(out = fopen(argv[2], "w")))
00105         ERRMSG("Cannot create file!");
00106
00107     /* Write header... */
00108     fprintf(out,
00109           "# $1 = time [s]\n"
00110           "# $2 = time difference [s]\n"
00111           "# $3 = absolute horizontal distance (%s) [km]\n"
00112           "# $4 = relative horizontal distance (%s) [%%]\n"
00113           "# $5 = absolute vertical distance (%s) [km]\n"
00114           "# $6 = relative vertical distance (%s) [%%]\n",
00115           argv[3], argv[3], argv[3], argv[3]);
00116     for (iq = 0; iq < ctl.nq; iq++)
00117         fprintf(out,

```

```

00118         "# %d = %s absolute difference (%s) [%s]\n"
00119         "# %d = %s relative difference (%s) [%s]\n",
00120         7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00121         8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00122     fprintf(out, "# %d = number of particles\n\n", 7 + 2 * ctl.nq);
00123
00124     /* Loop over file pairs... */
00125     for (f = 4; f < argc; f += 2) {
00126
00127         /* Read atmospheric data... */
00128         if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00129             continue;
00130
00131         /* Check if structs match... */
00132         if (atm1->np != atm2->np)
00133             ERRMSG("Different numbers of particles!");
00134
00135         /* Get time from filename... */
00136         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00137         year = atoi(tstr);
00138         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00139         mon = atoi(tstr);
00140         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00141         day = atoi(tstr);
00142         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00143         hour = atoi(tstr);
00144         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00145         min = atoi(tstr);
00146         time2jsec(year, mon, day, hour, min, 0, 0, &t);
00147
00148         /* Check time... */
00149         if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00150             || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00151             ERRMSG("Cannot read time from filename!");
00152
00153         /* Save initial time... */
00154         if (!init) {
00155             init = 1;
00156             t0 = t;
00157         }
00158
00159         /* Init... */
00160         np = 0;
00161         for (ip = 0; ip < atm1->np; ip++) {
00162             ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00163             for (iq = 0; iq < ctl.nq; iq++)
00164                 aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00165         }
00166
00167         /* Loop over air parcels... */
00168         for (ip = 0; ip < atm1->np; ip++) {
00169
00170             /* Check data... */
00171             if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00172                 continue;
00173
00174             /* Check ensemble index... */
00175             if (ctl.qnt_ens > 0
00176                 && (atm1->q[ctl.qnt_ens][ip] != ens
00177                     || atm2->q[ctl.qnt_ens][ip] != ens))
00178                 continue;
00179
00180             /* Check spatial range... */
00181             if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00182                 || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00183                 || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00184                 continue;
00185             if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00186                 || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00187                 || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00188                 continue;
00189
00190             /* Convert coordinates... */
00191             geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00192             geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00193             z1 = Z(atm1->p[ip]);
00194             z2 = Z(atm2->p[ip]);
00195
00196             /* Calculate absolute transport deviations... */
00197             ahtd[np] = DIST(x1, x2);
00198             avtd[np] = z1 - z2;
00199             for (iq = 0; iq < ctl.nq; iq++)
00200                 aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00201
00202             /* Calculate relative transport deviations... */
00203             if (f > 4) {
00204

```

```

00205     /* Get trajectory lengths... */
00206     geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00207     lh1[ip] += DIST(x0, x1);
00208     lv1[ip] += fabs(z1_old[ip] - z1);
00209
00210     geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00211     lh2[ip] += DIST(x0, x2);
00212     lv2[ip] += fabs(z2_old[ip] - z2);
00213
00214     /* Get relative transport deviations... */
00215     if (lh1[ip] + lh2[ip] > 0)
00216         rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00217     if (lv1[ip] + lv2[ip] > 0)
00218         rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00219 }
00220
00221 /* Get relative transport deviations... */
00222 for (iq = 0; iq < ctl.nq; iq++)
00223     rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00224     / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00225
00226 /* Save positions of air parcels... */
00227 lon1_old[ip] = atm1->lon[ip];
00228 lat1_old[ip] = atm1->lat[ip];
00229 z1_old[ip] = z1;
00230
00231 lon2_old[ip] = atm2->lon[ip];
00232 lat2_old[ip] = atm2->lat[ip];
00233 z2_old[ip] = z2;
00234
00235 /* Increment air parcel counter... */
00236 np++;
00237 }
00238
00239 /* Filter data... */
00240 if (zscore > 0 && np > 1) {
00241
00242     /* Get means and standard deviations of transport deviations... */
00243     size_t n = (size_t) np;
00244     double muh = gsl_stats_mean(ahtd, 1, n);
00245     double muv = gsl_stats_mean(avtd, 1, n);
00246     double sigh = gsl_stats_sd(ahtd, 1, n);
00247     double sigv = gsl_stats_sd(avtd, 1, n);
00248
00249     /* Filter data... */
00250     np = 0;
00251     for (size_t i = 0; i < n; i++)
00252         if (fabs((ahtd[i] - muh) / sigh) < zscore
00253             && fabs((avtd[i] - muv) / sigv) < zscore) {
00254             ahtd[np] = ahtd[i];
00255             rhtd[np] = rhtd[i];
00256             avtd[np] = avtd[i];
00257             rvtd[np] = rvtd[i];
00258             for (iq = 0; iq < ctl.nq; iq++) {
00259                 aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
00260                 rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
00261             }
00262             np++;
00263         }
00264 }
00265
00266 /* Get statistics... */
00267 if (strcmp(argv[3], "mean") == 0) {
00268     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00269     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00270     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00271     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00272     for (iq = 0; iq < ctl.nq; iq++) {
00273         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00274         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00275     }
00276 } else if (strcmp(argv[3], "stddev") == 0) {
00277     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00278     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00279     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00280     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00281     for (iq = 0; iq < ctl.nq; iq++) {
00282         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00283         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00284     }
00285 } else if (strcmp(argv[3], "min") == 0) {
00286     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00287     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00288     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00289     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00290     for (iq = 0; iq < ctl.nq; iq++) {
00291         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);

```



```

00292     rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00293 }
00294 } else if (strcasecmp(argv[3], "max") == 0) {
00295     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00296     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00297     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00298     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00299     for (iq = 0; iq < ctl.nq; iq++) {
00300         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00301         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00302     }
00303 } else if (strcasecmp(argv[3], "skew") == 0) {
00304     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00305     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00306     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00307     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00308     for (iq = 0; iq < ctl.nq; iq++) {
00309         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00310         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00311     }
00312 } else if (strcasecmp(argv[3], "kurt") == 0) {
00313     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00314     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00315     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00316     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00317     for (iq = 0; iq < ctl.nq; iq++) {
00318         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00319         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00320     }
00321 } else if (strcasecmp(argv[3], "absdev") == 0) {
00322     ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
00323     rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
00324     avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
00325     rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
00326     for (iq = 0; iq < ctl.nq; iq++) {
00327         aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
00328         rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
00329     }
00330 } else if (strcasecmp(argv[3], "median") == 0) {
00331     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00332     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00333     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00334     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00335     for (iq = 0; iq < ctl.nq; iq++) {
00336         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00337         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00338     }
00339 } else if (strcasecmp(argv[3], "mad") == 0) {
00340     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00341     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00342     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00343     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00344     for (iq = 0; iq < ctl.nq; iq++) {
00345         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00346         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00347     }
00348 } else
00349     ERRMSG("Unknown parameter!");
00350
00351 /* Write output... */
00352 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00353         ahtdm, rhtdm, avtdm, rvtdm);
00354 for (iq = 0; iq < ctl.nq; iq++) {
00355     fprintf(out, " ");
00356     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00357     fprintf(out, " ");
00358     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00359 }
00360 fprintf(out, " %d\n", np);
00361 }
00362
00363 /* Close file... */
00364 fclose(out);
00365
00366 /* Free... */
00367 free(atm1);
00368 free(atm2);
00369 free(lon1_old);
00370 free(lat1_old);
00371 free(zl_old);
00372 free(lh1);
00373 free(lv1);
00374 free(lon2_old);
00375 free(lat2_old);
00376 free(z2_old);
00377 free(lh2);
00378 free(lv2);

```

```

00379     free(ahtd);
00380     free(avtd);
00381     free(aqtd);
00382     free(rhtd);
00383     free(rvtd);
00384     free(rqtd);
00385     free(work);
00386
00387     return EXIT_SUCCESS;
00388 }

```

5.5 atm_init.c File Reference

```
#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](#).

```

00029     {
00030
00031     atm_t *atm;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1, t, z,
00038            lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m, vmr, bellrad;
00039
00040     int even, ip, irep, rep;
00041
00042     /* Allocate... */
00043     ALLOC(atm, atm_t, 1);
00044
00045     /* Check arguments... */
00046     if (argc < 3)
00047         ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049     /* Read control parameters... */
00050     read_ctl(argv[1], argc, argv, &ctl);
00051     t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052     t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053     dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054     z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055     z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056     dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);

```

```

00057 lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058 lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059 dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060 lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061 lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062 dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063 st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064 sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065 slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066 slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067 sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068 ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069 uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070 ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071 ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072 even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "0", NULL);
00073 rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074 m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075 vmr = scan_ctl(argv[1], argc, argv, "INIT_VMR", -1, "0", NULL);
00076 bellrad = scan_ctl(argv[1], argc, argv, "INIT_BELLRAD", -1, "0", NULL);
00077
00078 /* Initialize random number generator... */
00079 gsl_rng_env_setup();
00080 rng = gsl_rng_alloc(gsl_rng_default);
00081
00082 /* Create grid... */
00083 for (t = t0; t <= t1; t += dt)
00084     for (z = z0; z <= z1; z += dz)
00085         for (lon = lon0; lon <= lon1; lon += dlon)
00086             for (lat = lat0; lat <= lat1; lat += dlat)
00087                 for (irep = 0; irep < rep; irep++) {
00088
00089                     /* Set position... */
00090                     atm->time[atm->np]
00091                         = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00092                            + ut * (gsl_rng_uniform(rng) - 0.5));
00093                     atm->p[atm->np]
00094                         = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00095                            + uz * (gsl_rng_uniform(rng) - 0.5));
00096                     atm->lon[atm->np]
00097                         = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00098                            + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00099                            + ulon * (gsl_rng_uniform(rng) - 0.5));
00100                     do {
00101                         atm->lat[atm->np]
00102                             = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00103                                + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00104                                + ulat * (gsl_rng_uniform(rng) - 0.5));
00105                     } while (even && gsl_rng_uniform(rng) >
00106                             fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00107
00108                     /* Apply cosine bell (Williamson et al., 1992)... */
00109                     if (bellrad > 0) {
00110                         double x0[3], x1[3];
00111                         geo2cart(0.0, 0.5 * (lon0 + lon1), 0.5 * (lat0 + lat1), x0);
00112                         geo2cart(0.0, atm->lon[atm->np], atm->lat[atm->np], x1);
00113                         double rad = RE * acos(DOTP(x0, x1) / NORM(x0) / NORM(x1));
00114                         if (rad > bellrad)
00115                             continue;
00116                         if (ctl.qnt_m >= 0)
00117                             atm->q[ctl.qnt_m][atm->np] =
00118                                 0.5 * (1. + cos(M_PI * rad / bellrad));
00119                         if (ctl.qnt_vmr >= 0)
00120                             atm->q[ctl.qnt_vmr][atm->np] =
00121                                 0.5 * (1. + cos(M_PI * rad / bellrad));
00122                     }
00123
00124                     /* Set particle counter... */
00125                     if ((++atm->np) > NP)
00126                         ERRMSG("Too many particles!");
00127                 }
00128
00129 /* Check number of air parcels... */
00130 if (atm->np <= 0)
00131     ERRMSG("Did not create any air parcels!");
00132
00133 /* Initialize mass... */
00134 if (ctl.qnt_m >= 0 && bellrad <= 0)
00135     for (ip = 0; ip < atm->np; ip++)
00136         atm->q[ctl.qnt_m][ip] = m / atm->np;
00137
00138 /* Initialize volume mixing ratio... */
00139 if (ctl.qnt_vmr >= 0 && bellrad <= 0)
00140     for (ip = 0; ip < atm->np; ip++)
00141         atm->q[ctl.qnt_vmr][ip] = vmr;
00142
00143 /* Save data... */

```

```

00144     write_atm(argv[2], &ctl, atm, 0);
00145
00146     /* Free... */
00147     gsl_rng_free(rng);
00148     free(atm);
00149
00150     return EXIT_SUCCESS;
00151 }

```

5.6 atm_init.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     atm_t *atm;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1, t, z,
00038            lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m, vmr, bellrad;
00039
00040     int even, ip, irep, rep;
00041
00042     /* Allocate... */
00043     ALLOC(atm, atm_t, 1);
00044
00045     /* Check arguments... */
00046     if (argc < 3)
00047         ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049     /* Read control parameters... */
00050     read_ctl(argv[1], argc, argv, &ctl);
00051     t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052     t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053     dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054     z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055     z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056     dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057     lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058     lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059     dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060     lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061     lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062     dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063     st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064     sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065     slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066     slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067     sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068     ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069     uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070     ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071     ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072     even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "0", NULL);
00073     rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074     m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075     vmr = scan_ctl(argv[1], argc, argv, "INIT_VMR", -1, "0", NULL);
00076     bellrad = scan_ctl(argv[1], argc, argv, "INIT_BELLRAD", -1, "0", NULL);
00077
00078     /* Initialize random number generator... */

```

```

00079  gsl_rng_env_setup();
00080  rng = gsl_rng_alloc(gsl_rng_default);
00081
00082  /* Create grid... */
00083  for (t = t0; t <= t1; t += dt)
00084      for (z = z0; z <= z1; z += dz)
00085          for (lon = lon0; lon <= lon1; lon += dlon)
00086              for (lat = lat0; lat <= lat1; lat += dlat)
00087                  for (irep = 0; irep < rep; irep++) {
00088
00089                      /* Set position... */
00090                      atm->time[atm->np]
00091                          = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00092                             + ut * (gsl_rng_uniform(rng) - 0.5));
00093                      atm->p[atm->np]
00094                          = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00095                             + uz * (gsl_rng_uniform(rng) - 0.5));
00096                      atm->lon[atm->np]
00097                          = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00098                             + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00099                             + ulon * (gsl_rng_uniform(rng) - 0.5));
00100                      do {
00101                          atm->lat[atm->np]
00102                              = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00103                                 + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00104                                 + ulat * (gsl_rng_uniform(rng) - 0.5));
00105                      } while (even && gsl_rng_uniform(rng) >
00106                             fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00107
00108                      /* Apply cosine bell (Williamson et al., 1992)... */
00109                      if (bellrad > 0) {
00110                          double x0[3], x1[3];
00111                          geo2cart(0.0, 0.5 * (lon0 + lon1), 0.5 * (lat0 + lat1), x0);
00112                          geo2cart(0.0, atm->lon[atm->np], atm->lat[atm->np], x1);
00113                          double rad = RE * acos(DOTP(x0, x1) / NORM(x0) / NORM(x1));
00114                          if (rad > bellrad)
00115                              continue;
00116                          if (ctl.qnt_m >= 0)
00117                              atm->q[ctl.qnt_m][atm->np] =
00118                                  0.5 * (1. + cos(M_PI * rad / bellrad));
00119                          if (ctl.qnt_vmr >= 0)
00120                              atm->q[ctl.qnt_vmr][atm->np] =
00121                                  0.5 * (1. + cos(M_PI * rad / bellrad));
00122                      }
00123
00124                      /* Set particle counter... */
00125                      if ((++atm->np) > NP)
00126                          ERRMSG("Too many particles!");
00127                  }
00128
00129  /* Check number of air parcels... */
00130  if (atm->np <= 0)
00131      ERRMSG("Did not create any air parcels!");
00132
00133  /* Initialize mass... */
00134  if (ctl.qnt_m >= 0 && bellrad <= 0)
00135      for (ip = 0; ip < atm->np; ip++)
00136          atm->q[ctl.qnt_m][ip] = m / atm->np;
00137
00138  /* Initialize volume mixing ratio... */
00139  if (ctl.qnt_vmr >= 0 && bellrad <= 0)
00140      for (ip = 0; ip < atm->np; ip++)
00141          atm->q[ctl.qnt_vmr][ip] = vmr;
00142
00143  /* Save data... */
00144  write_atm(argv[2], &ctl, atm, 0);
00145
00146  /* Free... */
00147  gsl_rng_free(rng);
00148  free(atm);
00149
00150  return EXIT_SUCCESS;
00151 }

```

5.7 atm_select.c File Reference

```
#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](#).

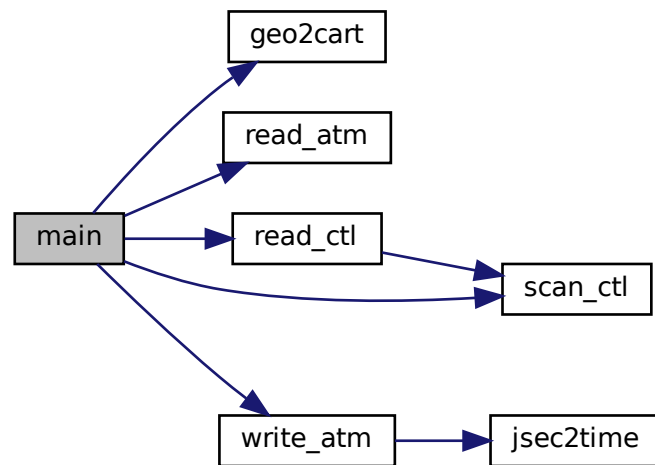
```
00029     {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm, *atm2;
00034
00035     double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
00036           x1[3];
00037
00038     int f, ip, ip0, ip1, iq, stride;
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042     ALLOC(atm2, atm_t, 1);
00043
00044     /* Check arguments... */
00045     if (argc < 4)
00046         ERRMSG("Give parameters: <ctl> <atm_select> <atm1> [<atm2> ...]");
00047
00048     /* Read control parameters... */
00049     read_ctl(argv[1], argc, argv, &ctl);
00050     stride =
00051         (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);
00052     ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00053     ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00054     t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00055     t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00056     p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00057     p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00058     lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00059     lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00060     lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00061     lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00062     r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00063     r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00064     rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00065     rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00066
00067     /* Get Cartesian coordinates... */
00068     geo2cart(0, rlon, rlat, x0);
00069
00070     /* Loop over files... */
00071     for (f = 3; f < argc; f++) {
00072
00073         /* Read atmospheric data... */
00074         if (!read_atm(argv[f], &ctl, atm))
00075             continue;
00076
00077         /* Adjust range of air parcels... */
00078         if (ip0 < 0)
00079             ip0 = 0;
00080         ip0 = GSL_MIN(ip0, atm->np - 1);
00081         if (ip1 < 0)
00082             ip1 = atm->np - 1;
00083         ip1 = GSL_MIN(ip1, atm->np - 1);
00084         if (ip1 < ip0)
00085             ip1 = ip0;
00086
00087         /* Loop over air parcels... */
00088         for (ip = ip0; ip <= ip1; ip += stride) {
```

```

00089
00090     /* Check time... */
00091     if (t0 != t1)
00092         if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00093             || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00094         continue;
00095
00096     /* Check vertical distance... */
00097     if (p0 != p1)
00098         if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00099             || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00100         continue;
00101
00102     /* Check longitude... */
00103     if (lon0 != lon1)
00104         if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00105             || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00106         continue;
00107
00108     /* Check latitude... */
00109     if (lat0 != lat1)
00110         if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00111             || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00112         continue;
00113
00114     /* Check horizontal distance... */
00115     if (r0 != r1) {
00116         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00117         r = DIST(x0, x1);
00118         if ((r1 > r0 && (r < r0 || r > r1))
00119             || (r1 < r0 && (r < r0 && r > r1)))
00120             continue;
00121     }
00122
00123     /* Copy data... */
00124     atm2->time[atm2->np] = atm->time[ip];
00125     atm2->p[atm2->np] = atm->p[ip];
00126     atm2->lon[atm2->np] = atm->lon[ip];
00127     atm2->lat[atm2->np] = atm->lat[ip];
00128     for (iq = 0; iq < ctl.nq; iq++)
00129         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00130     if (++atm2->np > NP)
00131         ERRMSG("Too many air parcels!");
00132 }
00133 }
00134
00135 /* Close file... */
00136 write_atm(argv[2], &ctl, atm2, 0);
00137
00138 /* Free... */
00139 free(atm);
00140 free(atm2);
00141
00142 return EXIT_SUCCESS;
00143 }

```

Here is the call graph for this function:



5.8 atm_select.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) 2013-2021 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  int main(
00028    int argc,
00029    char *argv[]) {
00030
00031    ctl_t ctl;
00032
00033    atm_t *atm, *atm2;
00034
00035    double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
00036           x1[3];
00037
00038    int f, ip, ip0, ip1, iq, stride;
00039
00040    /* Allocate... */
00041    ALLOC(atm, atm_t, 1);
00042    ALLOC(atm2, atm_t, 1);
00043
00044    /* Check arguments... */
00045    if (argc < 4)
00046        ERRMSG("Give parameters: <ctl> <atm_select> <atm1> [<atm2> ...]");
00047
00048    /* Read control parameters... */
00049    read_ctl(argv[1], argc, argv, &ctl);
00050    stride =
00051        (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);

```



```

00052 ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00053 ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00054 t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00055 t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00056 p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00057 p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00058 lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00059 lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00060 lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00061 lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00062 r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00063 r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00064 rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00065 rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00066
00067 /* Get Cartesian coordinates... */
00068 geo2cart(0, rlon, rlat, x0);
00069
00070 /* Loop over files... */
00071 for (f = 3; f < argc; f++) {
00072
00073     /* Read atmospheric data... */
00074     if (!read_atm(argv[f], &ctl, atm))
00075         continue;
00076
00077     /* Adjust range of air parcels... */
00078     if (ip0 < 0)
00079         ip0 = 0;
00080     ip0 = GSL_MIN(ip0, atm->np - 1);
00081     if (ip1 < 0)
00082         ip1 = atm->np - 1;
00083     ip1 = GSL_MIN(ip1, atm->np - 1);
00084     if (ip1 < ip0)
00085         ip1 = ip0;
00086
00087     /* Loop over air parcels... */
00088     for (ip = ip0; ip <= ip1; ip += stride) {
00089
00090         /* Check time... */
00091         if (t0 != t1)
00092             if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00093                 || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00094                 continue;
00095
00096         /* Check vertical distance... */
00097         if (p0 != p1)
00098             if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00099                 || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00100                 continue;
00101
00102         /* Check longitude... */
00103         if (lon0 != lon1)
00104             if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00105                 || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00106                 continue;
00107
00108         /* Check latitude... */
00109         if (lat0 != lat1)
00110             if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00111                 || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00112                 continue;
00113
00114         /* Check horizontal distance... */
00115         if (r0 != r1) {
00116             geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00117             r = DIST(x0, x1);
00118             if ((r1 > r0 && (r < r0 || r > r1))
00119                 || (r1 < r0 && (r < r0 && r > r1)))
00120                 continue;
00121         }
00122
00123         /* Copy data... */
00124         atm2->time[atm2->np] = atm->time[ip];
00125         atm2->p[atm2->np] = atm->p[ip];
00126         atm2->lon[atm2->np] = atm->lon[ip];
00127         atm2->lat[atm2->np] = atm->lat[ip];
00128         for (iq = 0; iq < ctl.nq; iq++)
00129             atm2->q[iq][atm2->np] = atm->q[iq][ip];
00130         if ((++atm2->np) > NP)
00131             ERRMSG("Too many air parcels!");
00132     }
00133 }
00134
00135 /* Close file... */
00136 write_atm(argv[2], &ctl, atm2, 0);
00137
00138 /* Free... */

```

```

00139     free(atm);
00140     free(atm2);
00141
00142     return EXIT_SUCCESS;
00143 }

```

5.9 atm_split.c File Reference

```
#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](#).

```

00029     {
00030
00031     atm_t *atm, *atm2;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     FILE *in;
00038
00039     char kernel[LEN], line[LEN];
00040
00041     double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
00042            t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;
00043
00044     int i, ip, iq, iz, n, nz = 0;
00045
00046     /* Allocate... */
00047     ALLOC(atm, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049
00050     /* Check arguments... */
00051     if (argc < 4)
00052         ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00053
00054     /* Read control parameters... */
00055     read_ctl(argv[1], argc, argv, &ctl);
00056     n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00057     m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00058     dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00059     t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00060     t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00061     dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);

```

```

00062 z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00063 z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00064 dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00065 lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00066 lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00067 lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00068 lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00069 scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);
00070
00071 /* Init random number generator... */
00072 gsl_rng_env_setup();
00073 rng = gsl_rng_alloc(gsl_rng_default);
00074
00075 /* Read atmospheric data... */
00076 if (!read_atm(argv[2], &ctl, atm))
00077     ERRMSG("Cannot open file!");
00078
00079 /* Read kernel function... */
00080 if (kernel[0] != '-') {
00081
00082     /* Write info... */
00083     LOG(1, "Read kernel function: %s", kernel);
00084
00085     /* Open file... */
00086     if (!(in = fopen(kernel, "r")))
00087         ERRMSG("Cannot open file!");
00088
00089     /* Read data... */
00090     while (fgets(line, LEN, in))
00091         if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
00092             if ((++nz) >= GZ)
00093                 ERRMSG("Too many height levels!");
00094
00095     /* Close file... */
00096     fclose(in);
00097
00098     /* Normalize kernel function... */
00099     zmax = gsl_stats_max(kz, 1, (size_t) nz);
00100     zmin = gsl_stats_min(kz, 1, (size_t) nz);
00101     kmax = gsl_stats_max(kk, 1, (size_t) nz);
00102     kmin = gsl_stats_min(kk, 1, (size_t) nz);
00103     for (iz = 0; iz < nz; iz++)
00104         kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00105 }
00106
00107 /* Get total and maximum mass... */
00108 if (ctl.qnt_m >= 0)
00109     for (ip = 0; ip < atm->np; ip++) {
00110         mtot += atm->q[ctl.qnt_m][ip];
00111         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00112     }
00113 if (m > 0)
00114     mtot = m;
00115
00116 /* Loop over air parcels... */
00117 for (i = 0; i < n; i++) {
00118
00119     /* Select air parcel... */
00120     if (ctl.qnt_m >= 0)
00121         do {
00122             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00123         } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00124     else
00125         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00126
00127     /* Set time... */
00128     if (t1 > t0)
00129         atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00130     else
00131         atm2->time[atm2->np] = atm->time[ip]
00132             + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00133
00134     /* Set vertical position... */
00135     do {
00136         if (nz > 0) {
00137             do {
00138                 z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
00139                 iz = locate_irr(kz, nz, z);
00140                 k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00141             } while (gsl_rng_uniform(rng) > k);
00142             atm2->p[atm2->np] = P(z);
00143         } else if (z1 > z0)
00144             atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00145         else
00146             atm2->p[atm2->np] = atm->p[ip]
00147                 + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00148     } while (atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));

```

```

00149
00150     /* Set horizontal position... */
00151     if (lon1 > lon0 && lat1 > lat0) {
00152         atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00153         atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00154     } else {
00155         atm2->lon[atm2->np] = atm->lon[ip]
00156         + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00157         atm2->lat[atm2->np] = atm->lat[ip]
00158         + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dy, atm->lat[ip]) / 2.3548);
00159     }
00160
00161     /* Copy quantities... */
00162     for (iq = 0; iq < ctl.nq; iq++)
00163         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00164
00165     /* Adjust mass... */
00166     if (ctl.qnt_m >= 0)
00167         atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00168
00169     /* Increment particle counter... */
00170     if ((++atm2->np) > NP)
00171         ERRMSG("Too many air parcels!");
00172 }
00173
00174 /* Save data and close file... */
00175 write_atm(argv[3], &ctl, atm2, 0);
00176
00177 /* Free... */
00178 free(atm);
00179 free(atm2);
00180
00181 return EXIT_SUCCESS;
00182 }

```

5.10 atm_split.c

```

00001 /*
00002     This file is part of MPTRAC.
00003
00004     MPTRAC is free software: you can redistribute it and/or modify
00005     it under the terms of the GNU General Public License as published by
00006     the Free Software Foundation, either version 3 of the License, or
00007     (at your option) any later version.
00008
00009     MPTRAC is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU General Public License for more details.
00013
00014     You should have received a copy of the GNU General Public License
00015     along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017     Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 int main(
00023     int argc,
00024     char *argv[]) {
00025
00026     atm_t *atm, *atm2;
00027
00028     ctl_t ctl;
00029
00030     gsl_rng *rng;
00031
00032     FILE *in;
00033
00034     char kernel[LEN], line[LEN];
00035
00036     double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
00037         t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;
00038
00039     int i, ip, iq, iz, n, nz = 0;
00040
00041     /* Allocate... */
00042     ALLOC(atm, atm_t, 1);
00043     ALLOC(atm2, atm_t, 1);
00044
00045     /* Check arguments... */
00046     if (argc < 4)
00047         ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");

```

```

00053
00054 /* Read control parameters... */
00055 read_ctl(argv[1], argc, argv, &ctl);
00056 n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00057 m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00058 dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00059 t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00060 t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00061 dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00062 z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00063 z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00064 dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00065 lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00066 lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00067 lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00068 lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00069 scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);
00070
00071 /* Init random number generator... */
00072 gsl_rng_env_setup();
00073 rng = gsl_rng_alloc(gsl_rng_default);
00074
00075 /* Read atmospheric data... */
00076 if (!read_atm(argv[2], &ctl, atm))
00077     ERRMSG("Cannot open file!");
00078
00079 /* Read kernel function... */
00080 if (kernel[0] != '-') {
00081
00082     /* Write info... */
00083     LOG(1, "Read kernel function: %s", kernel);
00084
00085     /* Open file... */
00086     if (!(in = fopen(kernel, "r")))
00087         ERRMSG("Cannot open file!");
00088
00089     /* Read data... */
00090     while (fgets(line, LEN, in))
00091         if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
00092             if (++nz >= GZ)
00093                 ERRMSG("Too many height levels!");
00094
00095     /* Close file... */
00096     fclose(in);
00097
00098     /* Normalize kernel function... */
00099     zmax = gsl_stats_max(kz, 1, (size_t) nz);
00100     zmin = gsl_stats_min(kz, 1, (size_t) nz);
00101     kmax = gsl_stats_max(kk, 1, (size_t) nz);
00102     kmin = gsl_stats_min(kk, 1, (size_t) nz);
00103     for (iz = 0; iz < nz; iz++)
00104         kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00105 }
00106
00107 /* Get total and maximum mass... */
00108 if (ctl.qnt_m >= 0)
00109     for (ip = 0; ip < atm->np; ip++) {
00110         mtot += atm->q[ctl.qnt_m][ip];
00111         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00112     }
00113 if (m > 0)
00114     mtot = m;
00115
00116 /* Loop over air parcels... */
00117 for (i = 0; i < n; i++) {
00118
00119     /* Select air parcel... */
00120     if (ctl.qnt_m >= 0)
00121         do {
00122             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00123             while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00124         } else
00125             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00126
00127     /* Set time... */
00128     if (t1 > t0)
00129         atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00130     else
00131         atm2->time[atm2->np] = atm->time[ip]
00132             + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00133
00134     /* Set vertical position... */
00135     do {
00136         if (nz > 0) {
00137             do {
00138                 z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
00139                 iz = locate_irr(kz, nz, z);

```

```

00140         k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00141     } while (gsl_rng_uniform(rng) > k);
00142     atm2->p[atm2->np] = P(z);
00143 } else if (z1 > z0)
00144     atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00145 else
00146     atm2->p[atm2->np] = atm->p[ip]
00147     + DZ2DP(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00148 } while (atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));
00149
00150 /* Set horizontal position... */
00151 if (lon1 > lon0 && lat1 > lat0) {
00152     atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00153     atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00154 } else {
00155     atm2->lon[atm2->np] = atm->lon[ip]
00156     + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00157     atm2->lat[atm2->np] = atm->lat[ip]
00158     + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dx) / 2.3548);
00159 }
00160
00161 /* Copy quantities... */
00162 for (iq = 0; iq < ctl.nq; iq++)
00163     atm2->q[iq][atm2->np] = atm->q[iq][ip];
00164
00165 /* Adjust mass... */
00166 if (ctl.qnt_m >= 0)
00167     atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00168
00169 /* Increment particle counter... */
00170 if ((++atm2->np) > NP)
00171     ERRMSG("Too many air parcels!");
00172 }
00173
00174 /* Save data and close file... */
00175 write_atm(argv[3], &ctl, atm2, 0);
00176
00177 /* Free... */
00178 free(atm);
00179 free(atm2);
00180
00181 return EXIT_SUCCESS;
00182 }

```

5.11 atm_stat.c File Reference

```
#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> ...]");
00055
00056         /* Read control parameters... */
00057         read_ctl(argv[1], argc, argv, &ctl);
00058         ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059         p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060         p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061         lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062         lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063         lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064         lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066         /* Write info... */
00067         printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069         /* Create output file... */
00070         if (!(out = fopen(argv[2], "w")))
00071             ERRMSG("Cannot create file!");
00072
00073         /* Write header... */
00074         fprintf(out,
00075             "# $1 = time [s]\n"
00076             "# $2 = time difference [s]\n"
00077             "# $3 = altitude (%) [km]\n"
00078             "# $4 = longitude (%) [deg]\n"
00079             "# $5 = latitude (%) [deg]\n", argv[3], argv[3], argv[3]);
00080         for (iq = 0; iq < ctl.nq; iq++)
00081             fprintf(out, "# $qd = %s (%s) [%s]\n", iq + 6,
00082                 ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083         fprintf(out, "# $qd = number of particles\n\n", ctl.nq + 6);
00084
00085         /* Loop over files... */
00086         for (f = 4; f < argc; f++) {
00087
00088             /* Read atmospheric data... */
00089             if (!read_atm(argv[f], &ctl, atm)
00090                 continue;
00091
00092             /* Get time from filename... */
00093             sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094             year = atoi(tstr);
00095             sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096             mon = atoi(tstr);
00097             sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098             day = atoi(tstr);
00099             sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100             hour = atoi(tstr);
00101             sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102             min = atoi(tstr);
00103             time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105             /* Check time... */
00106             if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00107                 || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00108                 ERRMSG("Cannot read time from filename!");

```

```

00109
00110 /* Save initial time... */
00111 if (!init) {
00112     init = 1;
00113     t0 = t;
00114 }
00115
00116 /* Filter data... */
00117 atm_filt->np = 0;
00118 for (ip = 0; ip < atm->np; ip++) {
00119
00120     /* Check time... */
00121     if (!gsl_finite(atm->time[ip]))
00122         continue;
00123
00124     /* Check ensemble index... */
00125     if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00126         continue;
00127
00128     /* Check spatial range... */
00129     if (atm->p[ip] > p0 || atm->p[ip] < p1
00130         || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00131         || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00132         continue;
00133
00134     /* Save data... */
00135     atm_filt->time[atm_filt->np] = atm->time[ip];
00136     atm_filt->p[atm_filt->np] = atm->p[ip];
00137     atm_filt->lon[atm_filt->np] = atm->lon[ip];
00138     atm_filt->lat[atm_filt->np] = atm->lat[ip];
00139     for (iq = 0; iq < ctl.nq; iq++)
00140         atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00141     atm_filt->np++;
00142 }
00143
00144 /* Get heights... */
00145 for (ip = 0; ip < atm_filt->np; ip++)
00146     zs[ip] = Z(atm_filt->p[ip]);
00147
00148 /* Get statistics... */
00149 if (strcmp(argv[3], "mean") == 0) {
00150     zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00151     lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00152     latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00153     for (iq = 0; iq < ctl.nq; iq++)
00154         qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00155 } else if (strcmp(argv[3], "stddev") == 0) {
00156     zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00157     lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00158     latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00159     for (iq = 0; iq < ctl.nq; iq++)
00160         qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00161 } else if (strcmp(argv[3], "min") == 0) {
00162     zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00163     lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00164     latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00165     for (iq = 0; iq < ctl.nq; iq++)
00166         qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00167 } else if (strcmp(argv[3], "max") == 0) {
00168     zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00169     lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00170     latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00171     for (iq = 0; iq < ctl.nq; iq++)
00172         qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00173 } else if (strcmp(argv[3], "skew") == 0) {
00174     zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00175     lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00176     latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00177     for (iq = 0; iq < ctl.nq; iq++)
00178         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00179 } else if (strcmp(argv[3], "kurt") == 0) {
00180     zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00181     lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00182     latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00183     for (iq = 0; iq < ctl.nq; iq++)
00184         qm[iq] =
00185             gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00186 } else if (strcmp(argv[3], "median") == 0) {
00187     zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00188     lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00189     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00190     for (iq = 0; iq < ctl.nq; iq++)
00191         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00192 } else if (strcmp(argv[3], "absdev") == 0) {
00193     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00194     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00195     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);

```

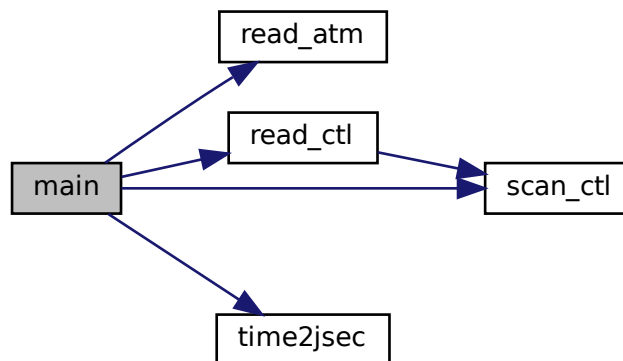


```

00196     for (iq = 0; iq < ctl.nq; iq++)
00197     {
00198         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00198     } else if (strcasecmp(argv[3], "mad") == 0) {
00199         zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00200         lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00201         latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00202         for (iq = 0; iq < ctl.nq; iq++)
00203             qm[iq] =
00204                 gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00205     } else
00206         ERRMSG("Unknown parameter!");
00207
00208     /* Write data... */
00209     fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00210     for (iq = 0; iq < ctl.nq; iq++) {
00211         fprintf(out, " ");
00212         fprintf(out, ctl.qnt_format[iq], qm[iq]);
00213     }
00214     fprintf(out, " %d\n", atm_filt->np);
00215 }
00216
00217 /* Close file... */
00218 fclose(out);
00219
00220 /* Free... */
00221 free(atm);
00222 free(atm_filt);
00223 free(work);
00224 free(zs);
00225
00226 return EXIT_SUCCESS;
00227 }

```

Here is the call graph for this function:



5.12 atm_stat.c

```

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

```

```

00017 Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm, *atm_filt;
00034
00035     FILE *out;
00036
00037     char tstr[LEN];
00038
00039     double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040         t, t0 = 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         NQ);
00049     ALLOC(zs, double,
00050         NQ);
00051
00052     /* Check arguments... */
00053     if (argc < 4)
00054         ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
00055
00056     /* Read control parameters... */
00057     read_ctl(argv[1], argc, argv, &ctl);
00058     ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059     p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060     p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061     lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062     lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063     lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064     lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066     /* Write info... */
00067     printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069     /* Create output file... */
00070     if (!(out = fopen(argv[2], "w")))
00071         ERRMSG("Cannot create file!");
00072
00073     /* Write header... */
00074     fprintf(out,
00075         "# $1 = time [s]\n"
00076         "# $2 = time difference [s]\n"
00077         "# $3 = altitude [%s] [km]\n"
00078         "# $4 = longitude [%s] [deg]\n"
00079         "# $5 = latitude [%s] [deg]\n", argv[3], argv[3], argv[3]);
00080     for (iq = 0; iq < ctl.nq; iq++)
00081         fprintf(out, "# %d = %s [%s] [%s]\n", iq + 6,
00082             ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083     fprintf(out, "# %d = number of particles\n\n", ctl.nq + 6);
00084
00085     /* Loop over files... */
00086     for (f = 4; f < argc; f++) {
00087
00088         /* Read atmospheric data... */
00089         if (!read_atm(argv[f], &ctl, atm))
00090             continue;
00091
00092         /* Get time from filename... */
00093         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094         year = atoi(tstr);
00095         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096         mon = atoi(tstr);
00097         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098         day = atoi(tstr);
00099         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100         hour = atoi(tstr);
00101         sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102         min = atoi(tstr);
00103         time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105         /* Check time... */
00106         if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00107             || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00108             ERRMSG("Cannot read time from filename!");

```

```

00109
00110 /* Save initial time... */
00111 if (!init) {
00112     init = 1;
00113     t0 = t;
00114 }
00115
00116 /* Filter data... */
00117 atm_filt->np = 0;
00118 for (ip = 0; ip < atm->np; ip++) {
00119
00120     /* Check time... */
00121     if (!gsl_finite(atm->time[ip]))
00122         continue;
00123
00124     /* Check ensemble index... */
00125     if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00126         continue;
00127
00128     /* Check spatial range... */
00129     if (atm->p[ip] > p0 || atm->p[ip] < p1
00130         || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00131         || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00132         continue;
00133
00134     /* Save data... */
00135     atm_filt->time[atm_filt->np] = atm->time[ip];
00136     atm_filt->p[atm_filt->np] = atm->p[ip];
00137     atm_filt->lon[atm_filt->np] = atm->lon[ip];
00138     atm_filt->lat[atm_filt->np] = atm->lat[ip];
00139     for (iq = 0; iq < ctl.nq; iq++)
00140         atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00141     atm_filt->np++;
00142 }
00143
00144 /* Get heights... */
00145 for (ip = 0; ip < atm_filt->np; ip++)
00146     zs[ip] = Z(atm_filt->p[ip]);
00147
00148 /* Get statistics... */
00149 if (strcmp(argv[3], "mean") == 0) {
00150     zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00151     lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00152     latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00153     for (iq = 0; iq < ctl.nq; iq++)
00154         qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00155 } else if (strcmp(argv[3], "stddev") == 0) {
00156     zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00157     lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00158     latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00159     for (iq = 0; iq < ctl.nq; iq++)
00160         qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00161 } else if (strcmp(argv[3], "min") == 0) {
00162     zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00163     lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00164     latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00165     for (iq = 0; iq < ctl.nq; iq++)
00166         qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00167 } else if (strcmp(argv[3], "max") == 0) {
00168     zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00169     lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00170     latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00171     for (iq = 0; iq < ctl.nq; iq++)
00172         qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00173 } else if (strcmp(argv[3], "skew") == 0) {
00174     zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00175     lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00176     latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00177     for (iq = 0; iq < ctl.nq; iq++)
00178         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00179 } else if (strcmp(argv[3], "kurt") == 0) {
00180     zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00181     lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00182     latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00183     for (iq = 0; iq < ctl.nq; iq++)
00184         qm[iq] =
00185             gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00186 } else if (strcmp(argv[3], "median") == 0) {
00187     zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00188     lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00189     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00190     for (iq = 0; iq < ctl.nq; iq++)
00191         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00192 } else if (strcmp(argv[3], "absdev") == 0) {
00193     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00194     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00195     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);

```

```

00196     for (iq = 0; iq < ctl.nq; iq++)
00197     { qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00198   } else if (strcasecmp(argv[3], "mad") == 0) {
00199     zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00200     lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00201     latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00202     for (iq = 0; iq < ctl.nq; iq++)
00203     { qm[iq] =
00204       gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00205   } else
00206     ERRMSG("Unknown parameter!");
00207
00208   /* Write data... */
00209   fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00210   for (iq = 0; iq < ctl.nq; iq++) {
00211     fprintf(out, " ");
00212     fprintf(out, ctl.qnt_format[iq], qm[iq]);
00213   }
00214   fprintf(out, " %d\n", atm_filt->np);
00215 }
00216
00217 /* Close file... */
00218 fclose(out);
00219
00220 /* Free... */
00221 free(atm);
00222 free(atm_filt);
00223 free(work);
00224 free(zs);
00225
00226 return EXIT_SUCCESS;
00227 }

```

5.13 day2doy.c File Reference

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

Functions

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

5.13.1 Detailed Description

Convert date to day of year.

Definition in file [day2doy.c](#).

5.13.2 Function Documentation

```

5.13.2.1 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:



5.14 day2doy.c

```

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

```

```
00043     day2doy(year, mon, day, &doy);
00044     printf("%d %d\n", year, doy);
00045
00046     return EXIT_SUCCESS;
00047 }
```

5.15 doy2day.c File Reference

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

Functions

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

5.15.1 Detailed Description

Convert day of year to date.

Definition in file [doy2day.c](#).

5.15.2 Function Documentation

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

Definition at line 27 of file [doy2day.c](#).

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

Here is the call graph for this function:



5.16 doy2day.c

```
00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013–2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 3)
00035         ERRMSG("Give parameters: <year> <doy>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     doy = atoi(argv[2]);
00040
00041     /* Convert... */
00042     doy2day(year, doy, &mon, &day);
00043     printf("%d %d %d\n", year, mon, day);
00044
00045     return EXIT_SUCCESS;
00046 }
```

5.17 jsec2time.c File Reference

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

Functions

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

5.17.1 Detailed Description

Convert Julian seconds to date.

Definition in file [jsec2time.c](#).

5.17.2 Function Documentation

```

5.17.2.1 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:



5.18 jsec2time.c

```

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

```



```

00043     jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
00044     printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046     return EXIT_SUCCESS;
00047 }

```

5.19 libtrac.c File Reference

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

Functions

- void [cart2geo](#) (double *x, double *z, double *lon, double *lat)
Convert Cartesian coordinates to geolocation.
- double [clim_hno3](#) (double t, double lat, double p)
Climatology of HNO3 volume mixing ratios.
- double [clim_oh](#) (double t, double lat, double p)
Climatology of OH number concentrations.
- double [clim_tropo](#) (double t, double lat)
Climatology of tropopause pressure.
- void [day2doy](#) (int year, int mon, int day, int *doy)
Get day of year from date.
- void [doy2day](#) (int year, int doy, int *mon, int *day)
Get date from day of year.
- void [geo2cart](#) (double z, double lon, double lat, double *x)
Convert geolocation to Cartesian coordinates.
- void [get_met](#) (ctl_t *ctl, double t, met_t **met0, met_t **met1)
Get meteorological data for given time step.
- void [get_met_help](#) (double t, int direct, char *metbase, double dt_met, char *filename)
Get meteorological 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 meteorological data.
- void [intpol_met_space_2d](#) (met_t *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteorological data.
- void [intpol_met_time_3d](#) (met_t *met0, float array0[EX][EY][EP], met_t *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void [intpol_met_time_2d](#) (met_t *met0, float array0[EX][EY], met_t *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void [jsec2time](#) (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)
Convert seconds to date.
- 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.
- int `read_atm` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
Read atmospheric data.
- void `read_ctl` (const char *filename, int argc, char *argv[], `ctl_t` *ctl)
Read control parameters.
- int `read_met` (`ctl_t` *ctl, char *filename, `met_t` *met)
Read meteorological data file.
- void `read_met_cape` (`met_t` *met)
Calculate convective available potential energy.
- void `read_met_cloud` (`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 meteorological 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 meteorological data.
- int `read_met_help_3d` (int ncid, char *varname, char *varname2, `met_t` *met, float dest[EX][EY][EP], float scl, int init)
Read and convert 3D variable from meteorological data file.
- int `read_met_help_2d` (int ncid, char *varname, char *varname2, `met_t` *met, float dest[EX][EY], float scl, int init)
Read and convert 2D variable from meteorological data file.
- void `read_met_levels` (int ncid, `ctl_t` *ctl, `met_t` *met)
Read meteorological data on vertical levels.
- void `read_met_ml2pl` (`ctl_t` *ctl, `met_t` *met, float var[EX][EY][EP])
Convert meteorological data from model levels to pressure levels.
- void `read_met_pbl` (`met_t` *met)
Calculate pressure of the boundary layer.
- void `read_met_periodic` (`met_t` *met)
Create meteorological data with periodic boundary conditions.
- void `read_met_pv` (`met_t` *met)
Calculate potential vorticity.
- void `read_met_sample` (`ctl_t` *ctl, `met_t` *met)
Downsampling of meteorological data.
- void `read_met_surface` (int ncid, `met_t` *met)
Read surface data.
- void `read_met_tropo` (`ctl_t` *ctl, `met_t` *met)
Calculate tropopause 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 r_p, double rho_p)
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.
- 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](#) (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_csi](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write CSI data.
- void [write_ens](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write ensemble data.
- void [write_grid](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write gridded data.
- void [write_prof](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write profile data.
- void [write_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.

5.19.1 Detailed Description

MPTRAC library definitions.

Definition in file [libtrac.c](#).

5.19.2 Function Documentation

5.19.2.1 [cart2geo\(\)](#) void [cart2geo](#) (
double * x,
double * z,
double * lon,
double * lat)

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file [libtrac.c](#).

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

```

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

```

Climatology of HNO3 volume mixing ratios.

Definition at line 295 of file [libtrac.c](#).

```

00298     {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Check latitude... */
00312     if (lat < clim_hno3_lats[0])
00313         lat = clim_hno3_lats[0];
00314     else if (lat > clim_hno3_lats[17])
00315         lat = clim_hno3_lats[17];
00316
00317     /* Get indices... */
00318     int isec = locate_irr(clim_hno3_secs, 12, sec);
00319     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00320     int ip = locate_irr(clim_hno3_ps, 10, p);
00321
00322     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00323     double aux00 = LIN(clim_hno3_ps[ip],
00324                        clim_hno3_var[isec][ilat][ip],
00325                        clim_hno3_ps[ip + 1],
00326                        clim_hno3_var[isec][ilat][ip + 1], p);
00327     double aux01 = LIN(clim_hno3_ps[ip],
00328                        clim_hno3_var[isec][ilat + 1][ip],
00329                        clim_hno3_ps[ip + 1],
00330                        clim_hno3_var[isec][ilat + 1][ip + 1], p);
00331     double aux10 = LIN(clim_hno3_ps[ip],
00332                        clim_hno3_var[isec + 1][ilat][ip],
00333                        clim_hno3_ps[ip + 1],
00334                        clim_hno3_var[isec + 1][ilat][ip + 1], p);
00335     double aux11 = LIN(clim_hno3_ps[ip],
00336                        clim_hno3_var[isec + 1][ilat + 1][ip],
00337                        clim_hno3_ps[ip + 1],
00338                        clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00339     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00340                clim_hno3_lats[ilat + 1], aux01, lat);
00341     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00342                clim_hno3_lats[ilat + 1], aux11, lat);
00343     aux00 = LIN(clim_hno3_secs[isec], aux00,
00344                clim_hno3_secs[isec + 1], aux11, sec);
00345
00346     /* Convert from ppb to ppv... */
00347     return GSL_MAX(1e-9 * aux00, 0.0);
00348 }

```

```

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

```

Climatology of OH number concentrations.

Definition at line 1331 of file [libtrac.c](#).

```

01334     {
01335
01336     /* Get seconds since begin of year... */
01337     double sec = FMOD(t, 365.25 * 86400.);
01338     while (sec < 0)
01339         sec += 365.25 * 86400.;
01340

```

```

01341  /* Check pressure... */
01342  if (p < clim_oh_ps[0])
01343      p = clim_oh_ps[0];
01344  else if (p > clim_oh_ps[33])
01345      p = clim_oh_ps[33];
01346
01347  /* Check latitude... */
01348  if (lat < clim_oh_lats[0])
01349      lat = clim_oh_lats[0];
01350  else if (lat > clim_oh_lats[17])
01351      lat = clim_oh_lats[17];
01352
01353  /* Get indices... */
01354  int isec = locate_irr(clim_oh_secs, 12, sec);
01355  int ilat = locate_reg(clim_oh_lats, 18, lat);
01356  int ip = locate_irr(clim_oh_ps, 34, p);
01357
01358  /* Interpolate OH climatology (Pommrich et al., 2014)... */
01359  double aux00 = LIN(clim_oh_ps[ip],
01360                    clim_oh_var[isec][ilat][ip],
01361                    clim_oh_ps[ip + 1],
01362                    clim_oh_var[isec][ilat][ip + 1], p);
01363  double aux01 = LIN(clim_oh_ps[ip],
01364                    clim_oh_var[isec][ilat + 1][ip],
01365                    clim_oh_ps[ip + 1],
01366                    clim_oh_var[isec][ilat + 1][ip + 1], p);
01367  double aux10 = LIN(clim_oh_ps[ip],
01368                    clim_oh_var[isec + 1][ilat][ip],
01369                    clim_oh_ps[ip + 1],
01370                    clim_oh_var[isec + 1][ilat][ip + 1], p);
01371  double aux11 = LIN(clim_oh_ps[ip],
01372                    clim_oh_var[isec + 1][ilat + 1][ip],
01373                    clim_oh_ps[ip + 1],
01374                    clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01375  aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01376  aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01377  aux00 = LIN(clim_oh_secs[isec], aux00, clim_oh_secs[isec + 1], aux11, sec);
01378  return GSL_MAX(1e6 * aux00, 0.0);
01379 }

```

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

Climatology of tropopause pressure.

Definition at line 1512 of file libtrac.c.

```

01514  {
01515
01516  /* Get seconds since begin of year... */
01517  double sec = FMOD(t, 365.25 * 86400.);
01518  while (sec < 0)
01519      sec += 365.25 * 86400.;
01520
01521  /* Get indices... */
01522  int isec = locate_irr(clim_tropo_secs, 12, sec);
01523  int ilat = locate_reg(clim_tropo_lats, 73, lat);
01524
01525  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01526  double p0 = LIN(clim_tropo_lats[ilat],
01527                clim_tropo_tps[isec][ilat],
01528                clim_tropo_lats[ilat + 1],
01529                clim_tropo_tps[isec][ilat + 1], lat);
01530  double p1 = LIN(clim_tropo_lats[ilat],
01531                clim_tropo_tps[isec + 1][ilat],
01532                clim_tropo_lats[ilat + 1],
01533                clim_tropo_tps[isec + 1][ilat + 1], lat);
01534  return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01535 }

```

Here is the call graph for this function:



5.19.2.5 day2doy() void day2doy (

```

    int year,
    int mon,
    int day,
    int * doy )

```

Get day of year from date.

Definition at line 1539 of file libtrac.c.

```

01543     {
01544
01545     const int
01546         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01547         d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01548
01549     /* Get day of year... */
01550     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01551         *doy = d0l[mon - 1] + day - 1;
01552     else
01553         *doy = d0[mon - 1] + day - 1;
01554 }

```

5.19.2.6 doy2day() void doy2day (

```

    int year,
    int doy,
    int * mon,
    int * day )

```

Get date from day of year.

Definition at line 1558 of file libtrac.c.

```

01562     {
01563
01564     const int
01565         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01566         d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01567
01568     int i;
01569
01570     /* Get month and day... */
01571     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01572         for (i = 11; i > 0; i--)
01573             if (d0l[i] <= doy)
01574                 break;
01575         *mon = i + 1;
01576         *day = doy - d0l[i] + 1;
01577     } else {
01578         for (i = 11; i > 0; i--)
01579             if (d0[i] <= doy)
01580                 break;
01581         *mon = i + 1;
01582         *day = doy - d0[i] + 1;
01583     }
01584 }

```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1588 of file libtrac.c.

```
01592     {
01593
01594     double radius = z + RE;
01595     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01596     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01597     x[2] = radius * sin(lat / 180. * M_PI);
01598 }
```

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

Get meteorological data for given time step.

Definition at line 1602 of file libtrac.c.

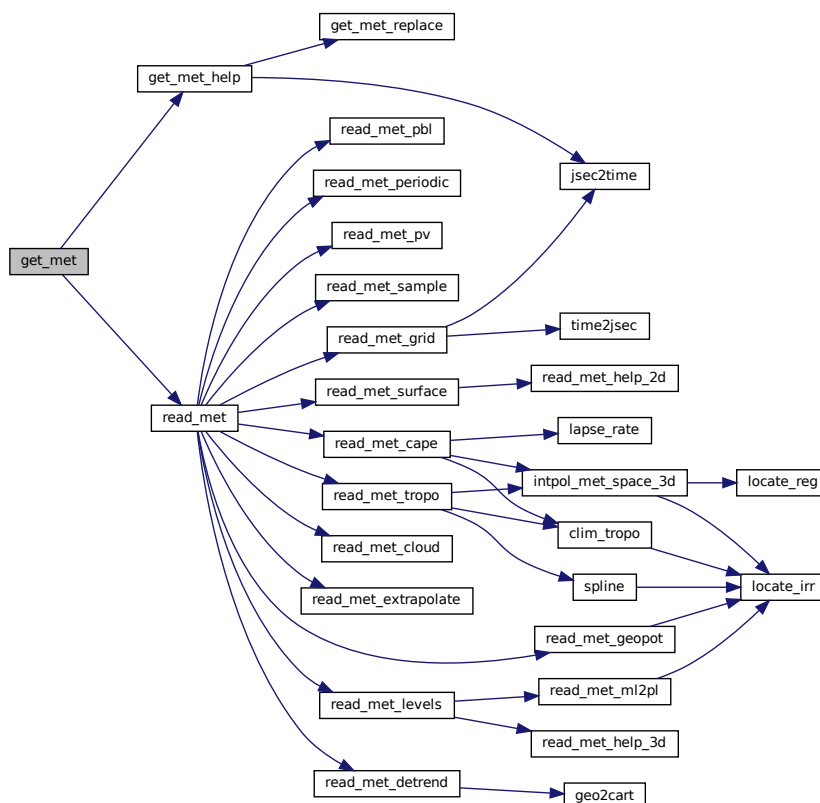
```
01606     {
01607
01608     static int init;
01609
01610     met_t *mets;
01611
01612     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01613
01614     /* Set timer... */
01615     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01616
01617     /* Init... */
01618     if (t == ctl->t_start || !init) {
01619         init = 1;
01620
01621         /* Read meteo data... */
01622         get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01623         if (!read_met(ctl, filename, *met0))
01624             ERRMSG("Cannot open file!");
01625
01626         get_met_help(t + 1.0 * ctl->direction, 1, ctl->metbase, ctl->dt_met,
01627                     filename);
01628         if (!read_met(ctl, filename, *met1))
01629             ERRMSG("Cannot open file!");
01630
01631         /* Update GPU... */
01632         #ifdef _OPENACC
01633             met_t *met0up = *met0;
01634             met_t *met1up = *met1;
01635             #pragma acc update device(met0up[:1],met1up[:1])
01636         #endif
01637
01638         /* Caching... */
01639         if (ctl->met_cache && t != ctl->t_stop) {
01640             get_met_help(t + 1.1 * ctl->dt_met * ctl->direction, ctl->direction,
01641                         ctl->metbase, ctl->dt_met, cachefile);
01642             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01643             LOG(1, "Caching: %s", cachefile);
01644             if (system(cmd) != 0)
01645                 WARN("Caching command failed!");
01646         }
01647     }
01648
01649     /* Read new data for forward trajectories... */
01650     if (t > (*met1)->time) {
01651
01652         /* Pointer swap... */
01653         mets = *met1;
```

```

01654     *met1 = *met0;
01655     *met0 = mets;
01656
01657     /* Read new meteo data... */
01658     get_met_help(t, 1, ctl->metbase, ctl->dt_met, filename);
01659     if (!read_met(ctl, filename, *met1))
01660         ERRMSG("Cannot open file!");
01661
01662     /* Update GPU... */
01663 #ifdef _OPENACC
01664     met_t *metlup = *met1;
01665     #pragma acc update device(metlup[:1])
01666 #endif
01667
01668     /* Caching... */
01669     if (ctl->met_cache && t != ctl->t_stop) {
01670         get_met_help(t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met, cachefile);
01671         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01672         LOG(1, "Caching: %s", cachefile);
01673         if (system(cmd) != 0)
01674             WARN("Caching command failed!");
01675     }
01676 }
01677
01678 /* Read new data for backward trajectories... */
01679 if (t < (*met0)->time) {
01680
01681     /* Pointer swap... */
01682     mets = *met1;
01683     *met1 = *met0;
01684     *met0 = mets;
01685
01686     /* Read new meteo data... */
01687     get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01688     if (!read_met(ctl, filename, *met0))
01689         ERRMSG("Cannot open file!");
01690
01691     /* Update GPU... */
01692 #ifdef _OPENACC
01693     met_t *met0up = *met0;
01694     #pragma acc update device(met0up[:1])
01695 #endif
01696
01697     /* Caching... */
01698     if (ctl->met_cache && t != ctl->t_stop) {
01699         get_met_help(t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met, cachefile);
01700         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01701         LOG(1, "Caching: %s", cachefile);
01702         if (system(cmd) != 0)
01703             WARN("Caching command failed!");
01704     }
01705 }
01706
01707 /* Check that grids are consistent... */
01708 if ((*met0)->nx != (*met1)->nx
01709     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01710     ERRMSG("Meteo grid dimensions do not match!");
01711 for (int ix = 0; ix < (*met0)->nx; ix++)
01712     if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01713         ERRMSG("Meteo grid longitudes do not match!");
01714 for (int iy = 0; iy < (*met0)->ny; iy++)
01715     if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01716         ERRMSG("Meteo grid latitudes do not match!");
01717 for (int ip = 0; ip < (*met0)->np; ip++)
01718     if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01719         ERRMSG("Meteo grid pressure levels do not match!");
01720 }

```


Here is the call graph for this function:



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

Get meteorological data for time step.

Definition at line 1724 of file libtrac.c.

```

01729 {
01730
01731     char repl[LEN];
01732
01733     double t6, r;
01734
01735     int year, mon, day, hour, min, sec;
01736
01737     /* Round time to fixed intervals... */
01738     if (direct == -1)
01739         t6 = floor(t / dt_met) * dt_met;
01740     else
01741         t6 = ceil(t / dt_met) * dt_met;
01742
01743     /* Decode time... */
01744     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01745
01746     /* Set filename... */

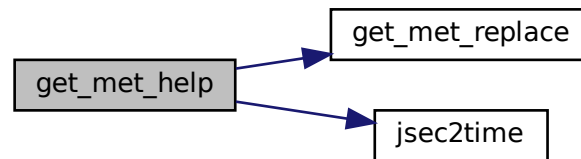
```

```

01747  sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01748  sprintf(repl, "%d", year);
01749  get_met_replace(filename, "YYYY", repl);
01750  sprintf(repl, "%02d", mon);
01751  get_met_replace(filename, "MM", repl);
01752  sprintf(repl, "%02d", day);
01753  get_met_replace(filename, "DD", repl);
01754  sprintf(repl, "%02d", hour);
01755  get_met_replace(filename, "HH", repl);
01756 }

```

Here is the call graph for this function:



5.19.2.10 get_met_replace() void get_met_replace (

```

    char * orig,
    char * search,
    char * repl )

```

Replace template strings in filename.

Definition at line 1760 of file libtrac.c.

```

01763  {
01764
01765  char buffer[LEN], *ch;
01766
01767  /* Iterate... */
01768  for (int i = 0; i < 3; i++) {
01769
01770  /* Replace sub-string... */
01771  if (!(ch = strstr(orig, search)))
01772  return;
01773  strncpy(buffer, orig, (size_t) (ch - orig));
01774  buffer[ch - orig] = 0;
01775  sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01776  orig[0] = 0;
01777  strcpy(orig, buffer);
01778  }
01779 }

```

5.19.2.11 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 meteorological data.

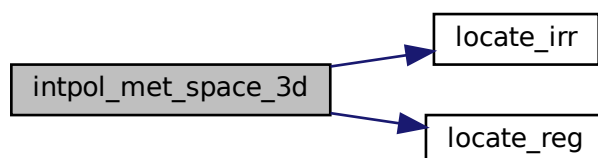
Definition at line 1783 of file libtrac.c.

```

1792     {
1793
1794     /* Initialize interpolation... */
1795     if (init) {
1796
1797         /* Check longitude... */
1798         if (met->lon[met->nx - 1] > 180 && lon < 0)
1799             lon += 360;
1800
1801         /* Get interpolation indices... */
1802         ci[0] = locate_irr(met->p, met->np, p);
1803         ci[1] = locate_reg(met->lon, met->nx, lon);
1804         ci[2] = locate_reg(met->lat, met->ny, lat);
1805
1806         /* Get interpolation weights... */
1807         cw[0] = (met->p[ci[0] + 1] - p)
1808             / (met->p[ci[0] + 1] - met->p[ci[0]]);
1809         cw[1] = (met->lon[ci[1] + 1] - lon)
1810             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
1811         cw[2] = (met->lat[ci[2] + 1] - lat)
1812             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
1813     }
1814
1815     /* Interpolate vertically... */
1816     double aux00 =
1817         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
1818         + array[ci[1]][ci[2]][ci[0] + 1];
1819     double aux01 =
1820         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
1821             array[ci[1]][ci[2] + 1][ci[0] + 1])
1822         + array[ci[1]][ci[2] + 1][ci[0] + 1];
1823     double aux10 =
1824         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
1825             array[ci[1] + 1][ci[2]][ci[0] + 1])
1826         + array[ci[1] + 1][ci[2]][ci[0] + 1];
1827     double aux11 =
1828         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
1829             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
1830         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
1831
1832     /* Interpolate horizontally... */
1833     aux00 = cw[2] * (aux00 - aux01) + aux01;
1834     aux11 = cw[2] * (aux10 - aux11) + aux11;
1835     *var = cw[1] * (aux00 - aux11) + aux11;
1836 }

```

Here is the call graph for this function:



```

5.19.2.12 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 meteorological data.

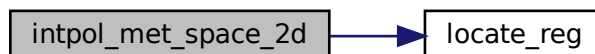
Definition at line 1841 of file libtrac.c.

```

01849     {
01850
01851     /* Initialize interpolation... */
01852     if (init) {
01853
01854         /* Check longitude... */
01855         if (met->lon[met->nx - 1] > 180 && lon < 0)
01856             lon += 360;
01857
01858         /* Get interpolation indices... */
01859         ci[1] = locate_reg(met->lon, met->nx, lon);
01860         ci[2] = locate_reg(met->lat, met->ny, lat);
01861
01862         /* Get interpolation weights... */
01863         cw[1] = (met->lon[ci[1] + 1] - lon)
01864             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01865         cw[2] = (met->lat[ci[2] + 1] - lat)
01866             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01867     }
01868
01869     /* Set variables... */
01870     double aux00 = array[ci[1]][ci[2]];
01871     double aux01 = array[ci[1]][ci[2] + 1];
01872     double aux10 = array[ci[1] + 1][ci[2]];
01873     double aux11 = array[ci[1] + 1][ci[2] + 1];
01874
01875     /* Interpolate horizontally... */
01876     if (isfinite(aux00) && isfinite(aux01)
01877         && isfinite(aux10) && isfinite(aux11)) {
01878         aux00 = cw[2] * (aux00 - aux01) + aux01;
01879         aux11 = cw[2] * (aux10 - aux11) + aux11;
01880         *var = cw[1] * (aux00 - aux11) + aux11;
01881     } else {
01882         if (cw[2] < 0.5) {
01883             if (cw[1] < 0.5)
01884                 *var = aux11;
01885             else
01886                 *var = aux01;
01887         } else {
01888             if (cw[1] < 0.5)
01889                 *var = aux10;
01890             else
01891                 *var = aux00;
01892         }
01893     }
01894 }

```

Here is the call graph for this function:



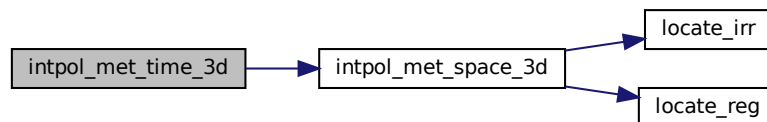
5.19.2.13 intpol_met_time_3d() void intpol_met_time_3d (
 met_t * met0,
 float array0[EX][EY][EP],
 met_t * met1,
 float array1[EX][EY][EP],
 double ts,
 double p,
 double lon,
 double lat,
 double * var,
 int * ci,
 double * cw,
 int init)

Temporal interpolation of meteorological data.

Definition at line 1898 of file libtrac.c.

```
01910     {
01911
01912     double var0, var1, wt;
01913
01914     /* Spatial interpolation... */
01915     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01916     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01917
01918     /* Get weighting factor... */
01919     wt = (met1->time - ts) / (met1->time - met0->time);
01920
01921     /* Interpolate... */
01922     *var = wt * (var0 - var1) + var1;
01923 }
```

Here is the call graph for this function:



5.19.2.14 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 meteorological data.

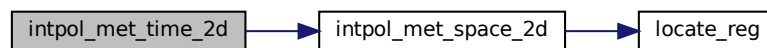
Definition at line 1927 of file libtrac.c.

```

01938     {
01939
01940     double var0, var1, wt;
01941
01942     /* Spatial interpolation... */
01943     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01944     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01945
01946     /* Get weighting factor... */
01947     wt = (met1->time - ts) / (met1->time - met0->time);
01948
01949     /* Interpolate... */
01950     if (isfinite(var0) && isfinite(var1))
01951         *var = wt * (var0 - var1) + var1;
01952     else if (wt < 0.5)
01953         *var = var1;
01954     else
01955         *var = var0;
01956 }

```

Here is the call graph for this function:



5.19.2.15 jsec2time() void jsec2time (

```

    double jsec,
    int * year,
    int * mon,
    int * day,
    int * hour,
    int * min,
    int * sec,
    double * remain )

```

Convert seconds to date.

Definition at line 1960 of file libtrac.c.

```

01968     {
01969
01970     struct tm t0, *t1;
01971
01972     t0.tm_year = 100;
01973     t0.tm_mon = 0;
01974     t0.tm_mday = 1;
01975     t0.tm_hour = 0;
01976     t0.tm_min = 0;
01977     t0.tm_sec = 0;
01978
01979     time_t jsec0 = (time_t) jsec + timegm(&t0);
01980     t1 = gmtime(&jsec0);
01981
01982     *year = t1->tm_year + 1900;
01983     *mon = t1->tm_mon + 1;
01984     *day = t1->tm_mday;
01985     *hour = t1->tm_hour;
01986     *min = t1->tm_min;
01987     *sec = t1->tm_sec;
01988     *remain = jsec - floor(jsec);
01989 }

```

5.19.2.16 lapse_rate() double lapse_rate (
double t,
double h2o)

Calculate moist adiabatic lapse rate.

Definition at line 1993 of file libtrac.c.

```
01995     {
01996
01997     /*
01998      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01999      and water vapor volume mixing ratio [1].
02000
02001      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
02002      */
02003
02004     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
02005
02006     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
02007 }
```

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

Find array index for irregular grid.

Definition at line 2011 of file libtrac.c.

```
02014     {
02015
02016     int ilo = 0;
02017     int ihi = n - 1;
02018     int i = (ihi + ilo) » 1;
02019
02020     if (xx[i] < xx[i + 1])
02021         while (ihi > ilo + 1) {
02022             i = (ihi + ilo) » 1;
02023             if (xx[i] > x)
02024                 ihi = i;
02025             else
02026                 ilo = i;
02027         } else
02028         while (ihi > ilo + 1) {
02029             i = (ihi + ilo) » 1;
02030             if (xx[i] <= x)
02031                 ihi = i;
02032             else
02033                 ilo = i;
02034         }
02035
02036     return ilo;
02037 }
```

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

Find array index for regular grid.

Definition at line 2041 of file libtrac.c.

```
02044     {
02045
02046     /* Calculate index... */
02047     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
```

```

02048
02049  /* Check range... */
02050  if (i < 0)
02051      return 0;
02052  else if (i > n - 2)
02053      return n - 2;
02054  else
02055      return i;
02056 }

```

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

Calculate NAT existence temperature.

Definition at line 2060 of file libtrac.c.

```

02063 {
02064
02065  /* Check water vapor vmr... */
02066  h2o = GSL_MAX(h2o, 0.1e-6);
02067
02068  /* Calculate T_NAT... */
02069  double p_hno3 = hno3 * p / 1.333224;
02070  double p_h2o = h2o * p / 1.333224;
02071  double a = 0.009179 - 0.00088 * log10(p_h2o);
02072  double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
02073  double c = -11397.0 / a;
02074  double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
02075  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
02076  if (x2 > 0)
02077      tnat = x2;
02078
02079  return tnat;
02080 }

```

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

Read atmospheric data.

Definition at line 2084 of file libtrac.c.

```

02087 {
02088
02089  FILE *in;
02090
02091  double t0;
02092
02093  int dimid, ncid, varid;
02094
02095  size_t nparts;
02096
02097  /* Set timer... */
02098  SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
02099
02100  /* Init... */
02101  atm->np = 0;
02102
02103  /* Write info... */
02104  LOG(1, "Read atmospheric data: %s", filename);
02105
02106  /* Read ASCII data... */
02107  if (ctl->atm_type == 0) {
02108
02109      /* Open file... */
02110      if (!(in = fopen(filename, "r"))) {

```



```

02111     WARN("File not found!");
02112     return 0;
02113 }
02114
02115 /* Read line... */
02116 char line[LEN];
02117 while (fgets(line, LEN, in)) {
02118
02119     /* Read data... */
02120     char *tok;
02121     TOK(line, tok, "%lg", atm->time[atm->np]);
02122     TOK(NULL, tok, "%lg", atm->p[atm->np]);
02123     TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02124     TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02125     for (int iq = 0; iq < ctl->nq; iq++)
02126         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02127
02128     /* Convert altitude to pressure... */
02129     atm->p[atm->np] = P(atm->p[atm->np]);
02130
02131     /* Increment data point counter... */
02132     if ((++atm->np) > NP)
02133         ERRMSG("Too many data points!");
02134 }
02135
02136 /* Close file... */
02137 fclose(in);
02138 }
02139
02140 /* Read binary data... */
02141 else if (ctl->atm_type == 1) {
02142
02143     /* Open file... */
02144     if (!(in = fopen(filename, "r")))
02145         return 0;
02146
02147     /* Read data... */
02148     FREAD(&atm->np, int,
02149          1,
02150          in);
02151     FREAD(atm->time, double,
02152          (size_t) atm->np,
02153          in);
02154     FREAD(atm->p, double,
02155          (size_t) atm->np,
02156          in);
02157     FREAD(atm->lon, double,
02158          (size_t) atm->np,
02159          in);
02160     FREAD(atm->lat, double,
02161          (size_t) atm->np,
02162          in);
02163     for (int iq = 0; iq < ctl->nq; iq++)
02164         FREAD(atm->q[iq], double,
02165              (size_t) atm->np,
02166              in);
02167
02168     /* Close file... */
02169     fclose(in);
02170 }
02171
02172 /* Read netCDF data... */
02173 else if (ctl->atm_type == 2) {
02174
02175     /* Open file... */
02176     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02177         return 0;
02178
02179     /* Get dimensions... */
02180     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02181     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02182     atm->np = (int) nparts;
02183     if (atm->np > NP)
02184         ERRMSG("Too many particles!");
02185
02186     /* Get time... */
02187     NC(nc_inq_varid(ncid, "time", &varid));
02188     NC(nc_get_var_double(ncid, varid, &t0));
02189     for (int ip = 0; ip < atm->np; ip++)
02190         atm->time[ip] = t0;
02191
02192     /* Read geolocations... */
02193     NC(nc_inq_varid(ncid, "PRESS", &varid));
02194     NC(nc_get_var_double(ncid, varid, atm->p));
02195     NC(nc_inq_varid(ncid, "LON", &varid));
02196     NC(nc_get_var_double(ncid, varid, atm->lon));
02197     NC(nc_inq_varid(ncid, "LAT", &varid));

```

```

02198     NC(nc_get_var_double(ncid, varid, atm->lat));
02199
02200     /* Read variables... */
02201     if (ctl->qnt_p >= 0)
02202         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02203             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02204     if (ctl->qnt_t >= 0)
02205         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02206             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02207     if (ctl->qnt_u >= 0)
02208         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02209             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02210     if (ctl->qnt_v >= 0)
02211         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02212             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02213     if (ctl->qnt_w >= 0)
02214         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02215             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02216     if (ctl->qnt_h2o >= 0)
02217         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02218             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02219     if (ctl->qnt_o3 >= 0)
02220         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02221             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02222     if (ctl->qnt_theta >= 0)
02223         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02224             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02225     if (ctl->qnt_pv >= 0)
02226         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02227             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02228
02229     /* Check data... */
02230     for (int ip = 0; ip < atm->np; ip++)
02231         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02232             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02233             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02234             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02235             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02236         atm->time[ip] = GSL_NAN;
02237         atm->p[ip] = GSL_NAN;
02238         atm->lon[ip] = GSL_NAN;
02239         atm->lat[ip] = GSL_NAN;
02240         for (int iq = 0; iq < ctl->nq; iq++)
02241             atm->q[iq][ip] = GSL_NAN;
02242     } else {
02243         if (ctl->qnt_h2o >= 0)
02244             atm->q[ctl->qnt_h2o][ip] *= 1.608;
02245         if (ctl->qnt_pv >= 0)
02246             atm->q[ctl->qnt_pv][ip] *= 1e6;
02247         if (atm->lon[ip] > 180)
02248             atm->lon[ip] -= 360;
02249     }
02250
02251     /* Close file... */
02252     NC(nc_close(ncid));
02253 }
02254
02255 /* Error... */
02256 else
02257     ERRMSG("Atmospheric data type not supported!");
02258
02259 /* Check number of points... */
02260 if (atm->np < 1)
02261     ERRMSG("Can not read any data!");
02262
02263 /* Write info... */
02264 double mini, maxi;
02265 LOG(2, "Number of particles: %d", atm->np);
02266 gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
02267 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
02268 gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
02269 LOG(2, "Altitude range: %g ... %g km", Z(mini), Z(maxi));
02270 LOG(2, "Pressure range: %g ... %g hPa", mini, maxi);
02271 gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
02272 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
02273 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
02274 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
02275 for (int iq = 0; iq < ctl->nq; iq++) {
02276     char msg[LEN];
02277     sprintf(msg, "Quantity %s range: %s ... %s %s",
02278             ctl->qnt_name[iq], ctl->qnt_format[iq],
02279             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
02280     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
02281     LOG(2, msg, mini, maxi);
02282 }
02283
02284 /* Return success... */

```

```
02285     return 1;
02286 }
```

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

Read control parameters.

Definition at line 2290 of file libtrac.c.

```
02294     {
02295
02296     /* Set timer... */
02297     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02298
02299     /* Write info... */
02300     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02301         "(executable: %s | version: %s | compiled: %s, %s)\n",
02302         argv[0], VERSION, __DATE__, __TIME__);
02303
02304     /* Initialize quantity indices... */
02305     ctl->qnt_ens = -1;
02306     ctl->qnt_stat = -1;
02307     ctl->qnt_m = -1;
02308     ctl->qnt_vmr = -1;
02309     ctl->qnt_r = -1;
02310     ctl->qnt_rho = -1;
02311     ctl->qnt_ps = -1;
02312     ctl->qnt_ts = -1;
02313     ctl->qnt_zs = -1;
02314     ctl->qnt_us = -1;
02315     ctl->qnt_vs = -1;
02316     ctl->qnt_pbl = -1;
02317     ctl->qnt_pt = -1;
02318     ctl->qnt_tt = -1;
02319     ctl->qnt_zt = -1;
02320     ctl->qnt_h2ot = -1;
02321     ctl->qnt_z = -1;
02322     ctl->qnt_p = -1;
02323     ctl->qnt_t = -1;
02324     ctl->qnt_u = -1;
02325     ctl->qnt_v = -1;
02326     ctl->qnt_w = -1;
02327     ctl->qnt_h2o = -1;
02328     ctl->qnt_o3 = -1;
02329     ctl->qnt_lwc = -1;
02330     ctl->qnt_iwc = -1;
02331     ctl->qnt_pct = -1;
02332     ctl->qnt_pcb = -1;
02333     ctl->qnt_cl = -1;
02334     ctl->qnt_plcl = -1;
02335     ctl->qnt_plfc = -1;
02336     ctl->qnt_pel = -1;
02337     ctl->qnt_cape = -1;
02338     ctl->qnt_cin = -1;
02339     ctl->qnt_hno3 = -1;
02340     ctl->qnt_oh = -1;
02341     ctl->qnt_psat = -1;
02342     ctl->qnt_psice = -1;
02343     ctl->qnt_pw = -1;
02344     ctl->qnt_sh = -1;
02345     ctl->qnt_rh = -1;
02346     ctl->qnt_rhice = -1;
02347     ctl->qnt_theta = -1;
02348     ctl->qnt_zeta = -1;
02349     ctl->qnt_tvirt = -1;
02350     ctl->qnt_lapse = -1;
02351     ctl->qnt_vh = -1;
02352     ctl->qnt_vz = -1;
02353     ctl->qnt_pv = -1;
02354     ctl->qnt_tdew = -1;
02355     ctl->qnt_tice = -1;
02356     ctl->qnt_tsts = -1;
02357     ctl->qnt_tnat = -1;
02358 }
```

```

02359  /* Read quantities... */
02360  ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02361  if (ctl->nq > NQ)
02362      ERRMSG("Too many quantities!");
02363  for (int iq = 0; iq < ctl->nq; iq++) {
02364
02365      /* Read quantity name and format... */
02366      scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02367      scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02368              ctl->qnt_format[iq]);
02369
02370      /* Try to identify quantity... */
02371      SET_QNT(qnt_ens, "ens", "-")
02372      SET_QNT(qnt_stat, "stat", "-")
02373      SET_QNT(qnt_m, "m", "kg")
02374      SET_QNT(qnt_vmr, "vmr", "ppv")
02375      SET_QNT(qnt_r, "r", "microns")
02376      SET_QNT(qnt_rho, "rho", "kg/m^3")
02377      SET_QNT(qnt_ps, "ps", "hPa")
02378      SET_QNT(qnt_ts, "ts", "K")
02379      SET_QNT(qnt_zs, "zs", "km")
02380      SET_QNT(qnt_us, "us", "m/s")
02381      SET_QNT(qnt_vs, "vs", "m/s")
02382      SET_QNT(qnt_pbl, "pbl", "hPa")
02383      SET_QNT(qnt_pt, "pt", "hPa")
02384      SET_QNT(qnt_tt, "tt", "K")
02385      SET_QNT(qnt_zt, "zt", "km")
02386      SET_QNT(qnt_h2ot, "h2ot", "ppv")
02387      SET_QNT(qnt_z, "z", "km")
02388      SET_QNT(qnt_p, "p", "hPa")
02389      SET_QNT(qnt_t, "t", "K")
02390      SET_QNT(qnt_u, "u", "m/s")
02391      SET_QNT(qnt_v, "v", "m/s")
02392      SET_QNT(qnt_w, "w", "hPa/s")
02393      SET_QNT(qnt_h2o, "h2o", "ppv")
02394      SET_QNT(qnt_o3, "o3", "ppv")
02395      SET_QNT(qnt_lwc, "lwc", "kg/kg")
02396      SET_QNT(qnt_iwc, "iwc", "kg/kg")
02397      SET_QNT(qnt_pct, "pct", "hPa")
02398      SET_QNT(qnt_pcb, "pcb", "hPa")
02399      SET_QNT(qnt_cl, "cl", "kg/m^2")
02400      SET_QNT(qnt_plcl, "plcl", "hPa")
02401      SET_QNT(qnt_plfc, "plfc", "hPa")
02402      SET_QNT(qnt_pel, "pel", "hPa")
02403      SET_QNT(qnt_cape, "cape", "J/kg")
02404      SET_QNT(qnt_cin, "cin", "J/kg")
02405      SET_QNT(qnt_hno3, "hno3", "ppv")
02406      SET_QNT(qnt_oh, "oh", "molec/cm^3")
02407      SET_QNT(qnt_psat, "psat", "hPa")
02408      SET_QNT(qnt_psize, "psize", "hPa")
02409      SET_QNT(qnt_pw, "pw", "hPa")
02410      SET_QNT(qnt_sh, "sh", "kg/kg")
02411      SET_QNT(qnt_rh, "rh", "%")
02412      SET_QNT(qnt_rhice, "rhice", "%")
02413      SET_QNT(qnt_theta, "theta", "K")
02414      SET_QNT(qnt_zeta, "zeta", "K")
02415      SET_QNT(qnt_tvirt, "tvirt", "K")
02416      SET_QNT(qnt_lapse, "lapse", "K/km")
02417      SET_QNT(qnt_vh, "vh", "m/s")
02418      SET_QNT(qnt_vz, "vz", "m/s")
02419      SET_QNT(qnt_pv, "pv", "PVU")
02420      SET_QNT(qnt_tdew, "tdew", "K")
02421      SET_QNT(qnt_tice, "tice", "K")
02422      SET_QNT(qnt_tsts, "tsts", "K")
02423      SET_QNT(qnt_tnat, "tnat", "K")
02424      scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02425  }
02426
02427  /* netCDF I/O parameters... */
02428  ctl->chunkszhint =
02429      (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02430                      NULL);
02431  ctl->read_mode =
02432      (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02433
02434  /* Time steps of simulation... */
02435  ctl->direction =
02436      (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02437  if (ctl->direction != -1 && ctl->direction != 1)
02438      ERRMSG("Set DIRECTION to -1 or 1!");
02439  ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02440  ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02441
02442  /* Meteorological data... */
02443  scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02444  ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02445  ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);

```

```

02446     ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02447     ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02448     if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02449         ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02450     ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02451     ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02452     ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02453     if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02454         ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02455     ctl->met_detrend =
02456         scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02457     ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02458     if (ctl->met_np > EP)
02459         ERRMSG("Too many levels!");
02460     for (int ip = 0; ip < ctl->met_np; ip++)
02461         ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02462     ctl->met_geopot_sx
02463         = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02464     ctl->met_geopot_sy
02465         = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02466     ctl->met_tropo =
02467         (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02468     if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02469         ERRMSG("Set MET_TROPO = 0 ... 5!");
02470     ctl->met_tropo_lapse =
02471         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02472     ctl->met_tropo_nlev =
02473         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02474     ctl->met_tropo_lapse_sep =
02475         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02476     ctl->met_tropo_nlev_sep =
02477         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02478             NULL);
02479     ctl->met_tropo_pv =
02480         scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02481     ctl->met_tropo_theta =
02482         scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02483     ctl->met_tropo_spline =
02484         (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02485     ctl->met_cloud =
02486         (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02487     if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02488         ERRMSG("Set MET_CLOUD = 0 ... 3!");
02489     ctl->met_dt_out =
02490         scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02491     ctl->met_cache =
02492         (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02493
02494     /* Isosurface parameters... */
02495     ctl->isosurf =
02496         (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02497     scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02498
02499     /* Advection parameters... */
02500     ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "0", NULL);
02501     ctl->reflect =
02502         (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02503
02504     /* Diffusion parameters... */
02505     ctl->turb_dx_trop =
02506         scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02507     ctl->turb_dx_strat =
02508         scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02509     ctl->turb_dz_trop =
02510         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02511     ctl->turb_dz_strat =
02512         scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02513     ctl->turb_mesox =
02514         scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02515     ctl->turb_mesoz =
02516         scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02517
02518     /* Convection... */
02519     ctl->conv_cape
02520         = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02521     ctl->conv_cin
02522         = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02523     ctl->conv_wmax
02524         = scan_ctl(filename, argc, argv, "CONV_WMAX", -1, "-999", NULL);
02525     ctl->conv_wcape
02526         = (int) scan_ctl(filename, argc, argv, "CONV_WCAPE", -1, "0", NULL);
02527     ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02528     ctl->conv_mix_bot
02529         = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02530     ctl->conv_mix_top
02531         = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02532

```

```

02533  /* Boundary conditions... */
02534  ctl->bound_mass =
02535      scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02536  ctl->bound_vmr =
02537      scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02538  ctl->bound_lat0 =
02539      scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02540  ctl->bound_lat1 =
02541      scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02542  ctl->bound_p0 =
02543      scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02544  ctl->bound_p1 =
02545      scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02546  ctl->bound_dps =
02547      scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02548
02549  /* Species parameters... */
02550  scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02551  if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02552      ctl->molmass = 120.907;
02553      ctl->wet_depo[2] = ctl->wet_depo[6] = 3e-5;
02554      ctl->wet_depo[3] = ctl->wet_depo[7] = 3500.0;
02555  } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02556      ctl->molmass = 137.359;
02557      ctl->wet_depo[2] = ctl->wet_depo[6] = 1.1e-4;
02558      ctl->wet_depo[3] = ctl->wet_depo[7] = 3300.0;
02559  } else if (strcasecmp(ctl->species, "CH4") == 0) {
02560      ctl->molmass = 16.043;
02561      ctl->oh_chem_reaction = 2;
02562      ctl->oh_chem[0] = 2.45e-12;
02563      ctl->oh_chem[1] = 1775;
02564      ctl->wet_depo[2] = ctl->wet_depo[6] = 1.4e-5;
02565      ctl->wet_depo[3] = ctl->wet_depo[7] = 1600.0;
02566  } else if (strcasecmp(ctl->species, "CO") == 0) {
02567      ctl->molmass = 28.01;
02568      ctl->oh_chem_reaction = 3;
02569      ctl->oh_chem[0] = 6.9e-33;
02570      ctl->oh_chem[1] = 2.1;
02571      ctl->oh_chem[2] = 1.1e-12;
02572      ctl->oh_chem[3] = -1.3;
02573      ctl->wet_depo[2] = ctl->wet_depo[6] = 9.7e-6;
02574      ctl->wet_depo[3] = ctl->wet_depo[7] = 1300.0;
02575  } else if (strcasecmp(ctl->species, "CO2") == 0) {
02576      ctl->molmass = 44.009;
02577      ctl->wet_depo[2] = ctl->wet_depo[6] = 3.3e-4;
02578      ctl->wet_depo[3] = ctl->wet_depo[7] = 2400.0;
02579  } else if (strcasecmp(ctl->species, "N2O") == 0) {
02580      ctl->molmass = 44.013;
02581      ctl->wet_depo[2] = ctl->wet_depo[6] = 2.4e-4;
02582      ctl->wet_depo[3] = ctl->wet_depo[7] = 2600.;
02583  } else if (strcasecmp(ctl->species, "NH3") == 0) {
02584      ctl->molmass = 17.031;
02585      ctl->oh_chem_reaction = 2;
02586      ctl->oh_chem[0] = 1.7e-12;
02587      ctl->oh_chem[1] = 710;
02588      ctl->wet_depo[2] = ctl->wet_depo[6] = 5.9e-1;
02589      ctl->wet_depo[3] = ctl->wet_depo[7] = 4200.0;
02590  } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02591      ctl->molmass = 63.012;
02592      ctl->wet_depo[2] = ctl->wet_depo[6] = 2.1e3;
02593      ctl->wet_depo[3] = ctl->wet_depo[7] = 8700.0;
02594  } else if (strcasecmp(ctl->species, "NO") == 0) {
02595      ctl->molmass = 30.006;
02596      ctl->oh_chem_reaction = 3;
02597      ctl->oh_chem[0] = 7.1e-31;
02598      ctl->oh_chem[1] = 2.6;
02599      ctl->oh_chem[2] = 3.6e-11;
02600      ctl->oh_chem[3] = 0.1;
02601      ctl->wet_depo[2] = ctl->wet_depo[6] = 1.9e-5;
02602      ctl->wet_depo[3] = ctl->wet_depo[7] = 1600.0;
02603  } else if (strcasecmp(ctl->species, "NO2") == 0) {
02604      ctl->molmass = 46.005;
02605      ctl->oh_chem_reaction = 3;
02606      ctl->oh_chem[0] = 1.8e-30;
02607      ctl->oh_chem[1] = 3.0;
02608      ctl->oh_chem[2] = 2.8e-11;
02609      ctl->oh_chem[3] = 0.0;
02610      ctl->wet_depo[2] = ctl->wet_depo[6] = 1.2e-4;
02611      ctl->wet_depo[3] = ctl->wet_depo[7] = 2400.0;
02612  } else if (strcasecmp(ctl->species, "O3") == 0) {
02613      ctl->molmass = 47.997;
02614      ctl->oh_chem_reaction = 2;
02615      ctl->oh_chem[0] = 1.7e-12;
02616      ctl->oh_chem[1] = 940;
02617      ctl->wet_depo[2] = ctl->wet_depo[6] = 1e-4;
02618      ctl->wet_depo[3] = ctl->wet_depo[7] = 2800.0;
02619  } else if (strcasecmp(ctl->species, "SF6") == 0) {

```

```

02620     ctl->molmass = 146.048;
02621     ctl->wet_depo[2] = ctl->wet_depo[6] = 2.4e-6;
02622     ctl->wet_depo[3] = ctl->wet_depo[7] = 3100.0;
02623 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02624     ctl->molmass = 64.066;
02625     ctl->oh_chem_reaction = 3;
02626     ctl->oh_chem[0] = 2.9e-31;
02627     ctl->oh_chem[1] = 4.1;
02628     ctl->oh_chem[2] = 1.7e-12;
02629     ctl->oh_chem[3] = -0.2;
02630     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.3e-2;
02631     ctl->wet_depo[3] = ctl->wet_depo[7] = 2900.0;
02632 } else {
02633     ctl->molmass =
02634         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02635     ctl->oh_chem_reaction =
02636         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02637     for (int ip = 0; ip < 4; ip++)
02638         ctl->oh_chem[ip] =
02639             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02640     for (int ip = 0; ip < 1; ip++)
02641         ctl->dry_depo[ip] =
02642             scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02643     for (int ip = 0; ip < 8; ip++)
02644         ctl->wet_depo[ip] =
02645             scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02646 }
02647 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02648 ctl->tdec_strat =
02649     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02650
02651 /* PSC analysis... */
02652 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02653 ctl->psc_hno3 =
02654     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02655
02656 /* Output of atmospheric data... */
02657 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02658 scan_ctl(filename, argc, argv, "ATM_GPFIL", -1, "-", ctl->atm_gpfile);
02659 ctl->atm_dt_out =
02660     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02661 ctl->atm_filter =
02662     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02663 ctl->atm_stride =
02664     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02665 ctl->atm_type =
02666     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02667
02668 /* Output of CSI data... */
02669 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02670 ctl->csi_dt_out =
02671     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02672 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02673 ctl->csi_obsmin =
02674     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02675 ctl->csi_modmin =
02676     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02677 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02678 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02679 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02680 ctl->csi_lon0 =
02681     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02682 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02683 ctl->csi_nx =
02684     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02685 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02686 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02687 ctl->csi_ny =
02688     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02689
02690 /* Output of ensemble data... */
02691 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02692
02693 /* Output of grid data... */
02694 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02695     ctl->grid_basename);
02696 scan_ctl(filename, argc, argv, "GRID_GPFIL", -1, "-", ctl->grid_gpfile);
02697 ctl->grid_dt_out =
02698     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02699 ctl->grid_sparse =
02700     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02701 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02702 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02703 ctl->grid_nz =
02704     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02705 ctl->grid_lon0 =
02706     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);

```

```

02707 ctl->grid_lon1 =
02708     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02709 ctl->grid_nx =
02710     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02711 ctl->grid_lat0 =
02712     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02713 ctl->grid_lat1 =
02714     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02715 ctl->grid_ny =
02716     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02717
02718 /* Output of profile data... */
02719 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02720         ctl->prof_basename);
02721 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02722 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02723 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02724 ctl->prof_nz =
02725     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02726 ctl->prof_lon0 =
02727     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02728 ctl->prof_lon1 =
02729     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02730 ctl->prof_nx =
02731     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02732 ctl->prof_lat0 =
02733     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02734 ctl->prof_lat1 =
02735     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02736 ctl->prof_ny =
02737     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02738
02739 /* Output of sample data... */
02740 scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02741         ctl->sample_basename);
02742 scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02743         ctl->sample_obsfile);
02744 ctl->sample_dx =
02745     scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02746 ctl->sample_dz =
02747     scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02748
02749 /* Output of station data... */
02750 scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02751         ctl->stat_basename);
02752 ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02753 ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02754 ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02755 ctl->stat_t0 =
02756     scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02757 ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02758 }

```

Here is the call graph for this function:



```

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

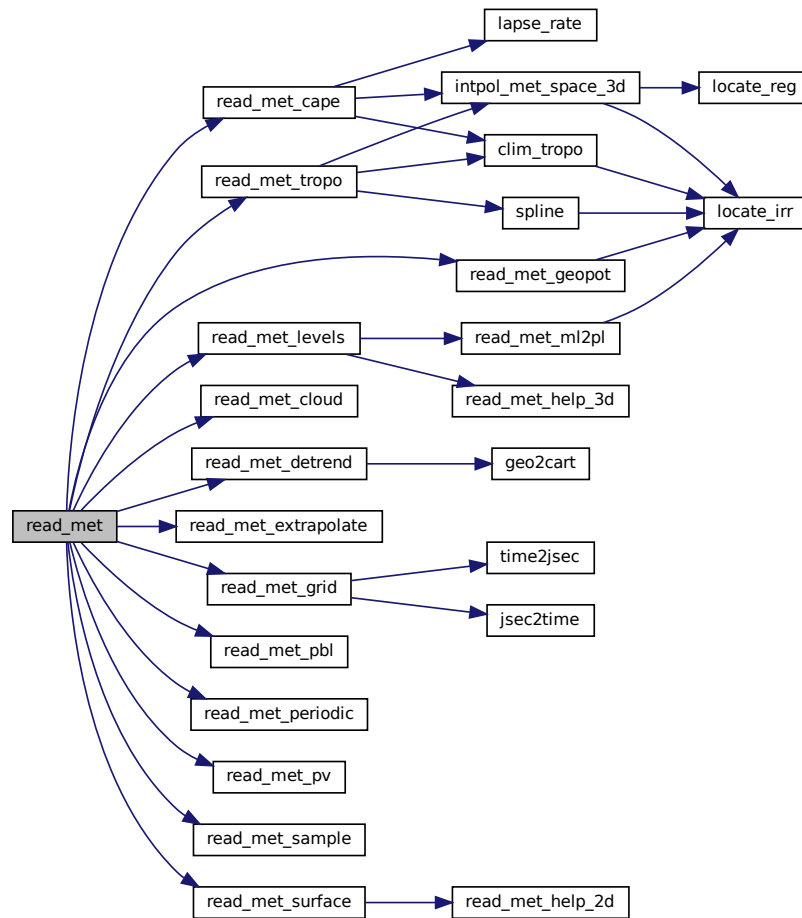
```

Read meteorological data file.

Definition at line 2762 of file libtrac.c.

```
02765         {
02766
02767     int ncid;
02768
02769     /* Write info... */
02770     LOG(1, "Read meteorological data: %s", filename);
02771
02772     /* Open netCDF file... */
02773     if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02774         NC_NOERR) {
02775         WARN("File not found!");
02776         return 0;
02777     }
02778
02779     /* Read coordinates of meteorological data... */
02780     read_met_grid(filename, ncid, ctl, met);
02781
02782     /* Read meteo data on vertical levels... */
02783     read_met_levels(ncid, ctl, met);
02784
02785     /* Extrapolate data for lower boundary... */
02786     read_met_extrapolate(met);
02787
02788     /* Read surface data... */
02789     read_met_surface(ncid, met);
02790
02791     /* Create periodic boundary conditions... */
02792     read_met_periodic(met);
02793
02794     /* Downsampling... */
02795     read_met_sample(ctl, met);
02796
02797     /* Calculate geopotential heights... */
02798     read_met_geopot(ctl, met);
02799
02800     /* Calculate potential vorticity... */
02801     read_met_pv(met);
02802
02803     /* Calculate boundary layer data... */
02804     read_met_pbl(met);
02805
02806     /* Calculate tropopause data... */
02807     read_met_tropo(ctl, met);
02808
02809     /* Calculate cloud properties... */
02810     read_met_cloud(met);
02811
02812     /* Calculate convective available potential energy... */
02813     read_metCAPE(met);
02814
02815     /* Detrending... */
02816     read_met_detrend(ctl, met);
02817
02818     /* Close file... */
02819     NC(nc_close(ncid));
02820
02821     /* Return success... */
02822     return 1;
02823 }
```

Here is the call graph for this function:



5.19.2.23 read_met_cape() void read_met_cape (
 met_t * met)

Calculate convective available potential energy.

Definition at line 2827 of file libtrac.c.

```

02828     {
02829
02830     /* Set timer... */
02831     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
02832     LOG(2, "Calculate CAPE...");
02833
02834     /* Vertical spacing (about 100 m)... */
02835     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
02836
02837     /* Loop over columns... */
02838     #pragma omp parallel for default(shared) collapse(2)
02839     for (int ix = 0; ix < met->nx; ix++)
02840     for (int iy = 0; iy < met->ny; iy++) {
02841
02842     /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
02843     int n = 0;
02844     double h2o = 0, t, theta = 0;
  
```

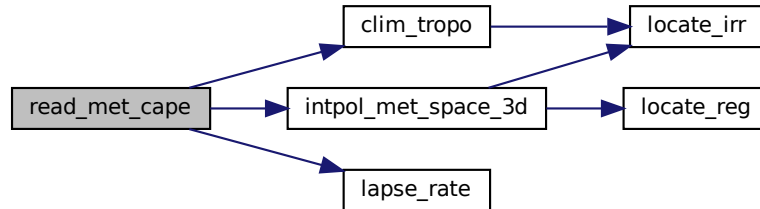
```

02845     double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
02846     double ptop = pbot - 50.;
02847     for (int ip = 0; ip < met->np; ip++) {
02848         if (met->p[ip] <= pbot) {
02849             theta += THETA(met->p[ip], met->t[ix][iy][ip]);
02850             h2o += met->h2o[ix][iy][ip];
02851             n++;
02852         }
02853         if (met->p[ip] < ptop && n > 0)
02854             break;
02855     }
02856     theta /= n;
02857     h2o /= n;
02858
02859     /* Cannot compute anything if water vapor is missing... */
02860     met->plcl[ix][iy] = GSL_NAN;
02861     met->plfc[ix][iy] = GSL_NAN;
02862     met->pel[ix][iy] = GSL_NAN;
02863     met->cape[ix][iy] = GSL_NAN;
02864     met->cin[ix][iy] = GSL_NAN;
02865     if (h2o <= 0)
02866         continue;
02867
02868     /* Find lifted condensation level (LCL)... */
02869     ptop = P(20.);
02870     pbot = met->ps[ix][iy];
02871     do {
02872         met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
02873         t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
02874         if (RH(met->plcl[ix][iy], t, h2o) > 100.)
02875             ptop = met->plcl[ix][iy];
02876         else
02877             pbot = met->plcl[ix][iy];
02878     } while (pbot - ptop > 0.1);
02879
02880     /* Calculate CIN up to LCL... */
02881     INTPOL_INIT;
02882     double dcape, dcape_old, dz, psat, h2o_env, t_env;
02883     double p = met->ps[ix][iy];
02884     met->cape[ix][iy] = met->cin[ix][iy] = 0;
02885     do {
02886         dz = dz0 * TVIRT(t, h2o);
02887         p /= pfac;
02888         t = theta / pow(1000. / p, 0.286);
02889         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02890                             &t_env, ci, cw, 1);
02891         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02892                             &h2o_env, ci, cw, 0);
02893         dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02894                 TVIRT(t_env, h2o_env) * dz;
02895         if (dcape < 0)
02896             met->cin[ix][iy] += fabsf((float) dcape);
02897     } while (p > met->plcl[ix][iy]);
02898
02899     /* Calculate level of free convection (LFC), equilibrium level (EL),
02900        and convective available potential energy (CAPE)... */
02901     dcape = 0;
02902     p = met->plcl[ix][iy];
02903     t = theta / pow(1000. / p, 0.286);
02904     ptop = 0.75 * clim_tropo(met->time, met->lat[iy]);
02905     do {
02906         dz = dz0 * TVIRT(t, h2o);
02907         p /= pfac;
02908         t -= lapse_rate(t, h2o) * dz;
02909         psat = PSAT(t);
02910         h2o = psat / (p - (1. - EPS) * psat);
02911         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02912                             &t_env, ci, cw, 1);
02913         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02914                             &h2o_env, ci, cw, 0);
02915         dcape_old = dcape;
02916         dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02917                 TVIRT(t_env, h2o_env) * dz;
02918         if (dcape > 0) {
02919             met->cape[ix][iy] += (float) dcape;
02920             if (!isfinite(met->plfc[ix][iy]))
02921                 met->plfc[ix][iy] = (float) p;
02922         } else if (dcape_old > 0)
02923             met->pel[ix][iy] = (float) p;
02924         if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
02925             met->cin[ix][iy] += fabsf((float) dcape);
02926     } while (p > ptop);
02927
02928     /* Check results... */
02929     if (!isfinite(met->plfc[ix][iy]))
02930         met->cin[ix][iy] = GSL_NAN;
02931 }

```

```
02932 }
```

Here is the call graph for this function:



5.19.2.24 read_met_cloud() void read_met_cloud (
 met_t * met)

Calculate cloud properties.

Definition at line 2936 of file libtrac.c.

```

02937     {
02938
02939     /* Set timer... */
02940     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
02941     LOG(2, "Calculate cloud data...");
02942
02943     /* Loop over columns... */
02944     #pragma omp parallel for default(shared) collapse(2)
02945     for (int ix = 0; ix < met->nx; ix++)
02946         for (int iy = 0; iy < met->ny; iy++) {
02947
02948         /* Init... */
02949         met->pct[ix][iy] = GSL_NAN;
02950         met->pcb[ix][iy] = GSL_NAN;
02951         met->cl[ix][iy] = 0;
02952
02953         /* Loop over pressure levels... */
02954         for (int ip = 0; ip < met->np - 1; ip++) {
02955
02956         /* Check pressure... */
02957         if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02958             continue;
02959
02960         /* Check ice water and liquid water content... */
02961         if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0) {
02962
02963         /* Get cloud top pressure ... */
02964         met->pct[ix][iy]
02965             = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
02966
02967         /* Get cloud bottom pressure ... */
02968         if (!isfinite(met->pcb[ix][iy]))
02969             met->pcb[ix][iy]
02970                 = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
02971         }
02972
02973         /* Get cloud water... */
02974         met->cl[ix][iy] += (float)
02975             (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02976                 + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02977              * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02978         }
02979     }
02980 }
```

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

Apply detrending method to temperature and winds.

Definition at line 2984 of file libtrac.c.

```

02986     {
02987
02988     met_t *help;
02989
02990     /* Check parameters... */
02991     if (ctl->met_detrend <= 0)
02992         return;
02993
02994     /* Set timer... */
02995     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
02996     LOG(2, "Detrend meteo data...");
02997
02998     /* Allocate... */
02999     ALLOC(help, met_t, 1);
03000
03001     /* Calculate standard deviation... */
03002     double sigma = ctl->met_detrend / 2.355;
03003     double tssq = 2. * SQR(sigma);
03004
03005     /* Calculate box size in latitude... */
03006     int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03007     sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03008
03009     /* Calculate background... */
03010     #pragma omp parallel for default(shared) collapse(2)
03011     for (int ix = 0; ix < met->nx; ix++) {
03012         for (int iy = 0; iy < met->ny; iy++) {
03013
03014             /* Calculate Cartesian coordinates... */
03015             double x0[3];
03016             geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03017
03018             /* Calculate box size in longitude... */
03019             int sx =
03020                 (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03021                     fabs(met->lon[1] - met->lon[0]));
03022             sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03023
03024             /* Init... */
03025             float wsum = 0;
03026             for (int ip = 0; ip < met->np; ip++) {
03027                 help->t[ix][iy][ip] = 0;
03028                 help->u[ix][iy][ip] = 0;
03029                 help->v[ix][iy][ip] = 0;
03030                 help->w[ix][iy][ip] = 0;
03031             }
03032
03033             /* Loop over neighboring grid points... */
03034             for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03035                 int ix3 = ix2;
03036                 if (ix3 < 0)
03037                     ix3 += met->nx;
03038                 else if (ix3 >= met->nx)
03039                     ix3 -= met->nx;
03040                 for (int iy2 = GSL_MAX(iy - sy, 0);
03041                     iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03042
03043                     /* Calculate Cartesian coordinates... */
03044                     double x1[3];
03045                     geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03046
03047                     /* Calculate weighting factor... */
03048                     float w = (float) exp(-DIST2(x0, x1) / tssq);
03049
03050                     /* Add data... */
03051                     wsum += w;
03052                     for (int ip = 0; ip < met->np; ip++) {
03053                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03054                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03055                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03056                         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03057                     }
03058                 }
03059             }
03060
03061             /* Normalize... */
03062             for (int ip = 0; ip < met->np; ip++) {

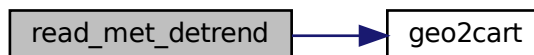
```

```

03063         help->t[ix][iy][ip] /= wsum;
03064         help->u[ix][iy][ip] /= wsum;
03065         help->v[ix][iy][ip] /= wsum;
03066         help->w[ix][iy][ip] /= wsum;
03067     }
03068 }
03069 }
03070
03071 /* Subtract background... */
03072 #pragma omp parallel for default(shared) collapse(3)
03073 for (int ix = 0; ix < met->nx; ix++)
03074     for (int iy = 0; iy < met->ny; iy++)
03075         for (int ip = 0; ip < met->np; ip++) {
03076             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03077             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03078             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03079             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03080         }
03081
03082 /* Free... */
03083 free(help);
03084 }

```

Here is the call graph for this function:



5.19.2.26 read_met_extrapolate() void read_met_extrapolate (
met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 3088 of file libtrac.c.

```

03089     {
03090
03091         /* Set timer... */
03092         SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03093         LOG(2, "Extrapolate meteo data...");
03094
03095         /* Loop over columns... */
03096         #pragma omp parallel for default(shared) collapse(2)
03097         for (int ix = 0; ix < met->nx; ix++)
03098             for (int iy = 0; iy < met->ny; iy++) {
03099
03100                 /* Find lowest valid data point... */
03101                 int ip0;
03102                 for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03103                     if (!isfinite(met->t[ix][iy][ip0])
03104                         || !isfinite(met->u[ix][iy][ip0])
03105                         || !isfinite(met->v[ix][iy][ip0])
03106                         || !isfinite(met->w[ix][iy][ip0]))
03107                         break;
03108
03109                 /* Extrapolate... */
03110                 for (int ip = ip0; ip >= 0; ip--) {
03111                     met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03112                     met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03113                     met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03114                     met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03115                     met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03116                     met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03117                     met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03118                     met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03119                 }
03120             }
03121 }

```

5.19.2.27 read_met_geopot() void read_met_geopot (

```

    ctl_t * ctl,
    met_t * met )

```

Calculate geopotential heights.

Definition at line 3125 of file libtrac.c.

```

3127     {
3128
3129         static float help[EP][EX][EY];
3130
3131         double logp[EP];
3132
3133         int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
3134
3135         /* Set timer... */
3136         SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
3137         LOG(2, "Calculate geopotential heights...");
3138
3139         /* Calculate log pressure... */
3140         #pragma omp parallel for default(shared)
3141         for (int ip = 0; ip < met->np; ip++)
3142             logp[ip] = log(met->p[ip]);
3143
3144         /* Apply hydrostatic equation to calculate geopotential heights... */
3145         #pragma omp parallel for default(shared) collapse(2)
3146         for (int ix = 0; ix < met->nx; ix++)
3147             for (int iy = 0; iy < met->ny; iy++) {
3148
3149                 /* Get surface height and pressure... */
3150                 double zs = met->zs[ix][iy];
3151                 double lnps = log(met->ps[ix][iy]);
3152
3153                 /* Get temperature and water vapor vmr at the surface... */
3154                 int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
3155                 double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
3156                                met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
3157                 double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
3158                                met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
3159
3160                 /* Upper part of profile... */
3161                 met->z[ix][iy][ip0 + 1]
3162                     = (float) (zs +
3163                               ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
3164                                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
3165                 for (int ip = ip0 + 2; ip < met->np; ip++)
3166                     met->z[ix][iy][ip]
3167                         = (float) (met->z[ix][iy][ip - 1] +
3168                                   ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
3169                                         met->h2o[ix][iy][ip - 1], logp[ip],
3170                                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
3171
3172                 /* Lower part of profile... */
3173                 met->z[ix][iy][ip0]
3174                     = (float) (zs +
3175                               ZDIFF(lnps, ts, h2os, logp[ip0],
3176                                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
3177                 for (int ip = ip0 - 1; ip >= 0; ip--)
3178                     met->z[ix][iy][ip]
3179                         = (float) (met->z[ix][iy][ip + 1] +
3180                                   ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
3181                                         met->h2o[ix][iy][ip + 1], logp[ip],
3182                                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
3183             }
3184
3185         /* Check control parameters... */
3186         if (dx == 0 || dy == 0)
3187             return;
3188
3189         /* Default smoothing parameters... */
3190         if (dx < 0 || dy < 0) {
3191             if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
3192                 dx = 3;
3193                 dy = 2;
3194             } else {
3195                 dx = 6;
3196                 dy = 4;
3197             }
3198         }
3199
3200         /* Calculate weights for smoothing... */
3201         float ws[dx + 1][dy + 1];
3202         #pragma omp parallel for default(shared) collapse(2)
3203         for (int ix = 0; ix <= dx; ix++)

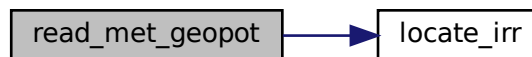
```

```

03204     for (int iy = 0; iy < dy; iy++)
03205         ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03206             * (1.0f - (float) iy / (float) dy);
03207
03208     /* Copy data... */
03209 #pragma omp parallel for default(shared) collapse(3)
03210     for (int ix = 0; ix < met->nx; ix++)
03211         for (int iy = 0; iy < met->ny; iy++)
03212             for (int ip = 0; ip < met->np; ip++)
03213                 help[ip][ix][iy] = met->z[ix][iy][ip];
03214
03215     /* Horizontal smoothing... */
03216 #pragma omp parallel for default(shared) collapse(3)
03217     for (int ip = 0; ip < met->np; ip++)
03218         for (int ix = 0; ix < met->nx; ix++)
03219             for (int iy = 0; iy < met->ny; iy++) {
03220                 float res = 0, wsum = 0;
03221                 int iy0 = GSL_MAX(iy - dy + 1, 0);
03222                 int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03223                 for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03224                     int ix3 = ix2;
03225                     if (ix3 < 0)
03226                         ix3 += met->nx;
03227                     else if (ix3 >= met->nx)
03228                         ix3 -= met->nx;
03229                     for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03230                         if (isfinite(help[ip][ix3][iy2])) {
03231                             float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03232                             res += w * help[ip][ix3][iy2];
03233                             wsum += w;
03234                         }
03235                 }
03236                 if (wsum > 0)
03237                     met->z[ix][iy][ip] = res / wsum;
03238                 else
03239                     met->z[ix][iy][ip] = GSL_NAN;
03240             }
03241 }

```

Here is the call graph for this function:



5.19.2.28 read_met_grid() void read_met_grid (

```

    char * filename,
    int ncid,
    ctl_t * ctl,
    met_t * met )

```

Read coordinates of meteorological data.

Definition at line 3245 of file libtrac.c.

```

03249     {
03250
03251         char levname[LEN], tstr[10];
03252
03253         double rtime, r2;
03254
03255         int dimid, varid, year2, mon2, day2, hour2, min2, sec2;
03256
03257         size_t np, nx, ny;
03258

```

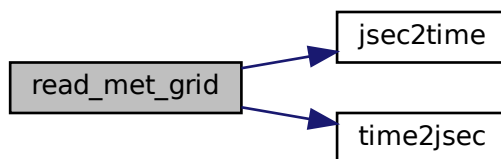


```

03259  /* Set timer... */
03260  SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03261  LOG(2, "Read meteo grid information...");
03262
03263  /* Get time from filename... */
03264  sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
03265  int year = atoi(tstr);
03266  sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
03267  int mon = atoi(tstr);
03268  sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
03269  int day = atoi(tstr);
03270  sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03271  int hour = atoi(tstr);
03272  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03273
03274  /* Check time... */
03275  if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03276      || day < 1 || day > 31 || hour < 0 || hour > 23)
03277      ERRMSG("Cannot read time from filename!");
03278  jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03279  LOG(2, "Time from filename: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03280      met->time, year2, mon2, day2, hour2, min2);
03281
03282  /* Check time information... */
03283  if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03284      NC(nc_get_var_double(ncid, varid, &rtime));
03285      if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
03286          WARN("Time information in meteo file does not match filename!");
03287  } else
03288      WARN("Time information in meteo file is missing!");
03289
03290  /* Get grid dimensions... */
03291  NC(nc_inq_dimid(ncid, "lon", &dimid));
03292  NC(nc_inq_dimlen(ncid, dimid, &nx));
03293  LOG(2, "Number of longitudes: %zu", nx);
03294  if (nx < 2 || nx > EX)
03295      ERRMSG("Number of longitudes out of range!");
03296
03297  NC(nc_inq_dimid(ncid, "lat", &dimid));
03298  NC(nc_inq_dimlen(ncid, dimid, &ny));
03299  LOG(2, "Number of latitudes: %zu", ny);
03300  if (ny < 2 || ny > EY)
03301      ERRMSG("Number of latitudes out of range!");
03302
03303  sprintf(levname, "lev");
03304  NC(nc_inq_dimid(ncid, levname, &dimid));
03305  NC(nc_inq_dimlen(ncid, dimid, &np));
03306  if (np == 1) {
03307      sprintf(levname, "lev_2");
03308      if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03309          sprintf(levname, "plev");
03310          nc_inq_dimid(ncid, levname, &dimid);
03311      }
03312      NC(nc_inq_dimlen(ncid, dimid, &np));
03313  }
03314  LOG(2, "Number of levels: %zu", np);
03315  if (np < 2 || np > EP)
03316      ERRMSG("Number of levels out of range!");
03317
03318  /* Store dimensions... */
03319  met->np = (int) np;
03320  met->nx = (int) nx;
03321  met->ny = (int) ny;
03322
03323  /* Read longitudes and latitudes... */
03324  NC(nc_inq_varid(ncid, "lon", &varid));
03325  NC(nc_get_var_double(ncid, varid, met->lon));
03326  LOG(2, "Longitudes: %g, %g ... %g deg",
03327      met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03328  NC(nc_inq_varid(ncid, "lat", &varid));
03329  NC(nc_get_var_double(ncid, varid, met->lat));
03330  LOG(2, "Latitudes: %g, %g ... %g deg",
03331      met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03332
03333  /* Read pressure levels... */
03334  if (ctl->met_np <= 0) {
03335      NC(nc_inq_varid(ncid, levname, &varid));
03336      NC(nc_get_var_double(ncid, varid, met->p));
03337      for (int ip = 0; ip < met->np; ip++)
03338          met->p[ip] /= 100.;
03339      LOG(2, "Altitude levels: %g, %g ... %g km",
03340          Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03341      LOG(2, "Pressure levels: %g, %g ... %g hPa",
03342          met->p[0], met->p[1], met->p[met->np - 1]);
03343  }
03344 }

```

Here is the call graph for this function:



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

Read and convert 3D variable from meteorological data file.

Definition at line 3348 of file [libtrac.c](#).

```

03355     {
03356
03357     char varsels[LEN];
03358
03359     float offset, scalfac;
03360
03361     int varid;
03362
03363     /* Check if variable exists... */
03364     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03365     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03366         WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03367         return 0;
03368     } else {
03369         sprintf(varsels, "%s", varname2);
03370     } else
03371         sprintf(varsels, "%s", varname);
03372
03373     /* Read packed data... */
03374     if (nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03375         && nc_get_att_float(ncid, varid, "scale_factor",
03376                             &scalfac) == NC_NOERR) {
03377
03378         /* Allocate... */
03379         short *help;
03380         ALLOC(help, short,
03381              EX * EY * EP);
03382
03383         /* Read fill value and missing value... */
03384         short fillval, missval;
03385         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03386             fillval = 0;
03387         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03388             missval = 0;
03389
03390         /* Write info... */
03391         LOG(2, "Read 3-D variable: %s "
03392             "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03393            varsels, fillval, missval, scalfac, offset);
03394     }
  
```

```

03395     /* Read data... */
03396     NC(nc_get_var_short(ncid, varid, help));
03397
03398     /* Copy and check data... */
03399 #pragma omp parallel for default(shared) num_threads(12)
03400     for (int ix = 0; ix < met->nx; ix++)
03401         for (int iy = 0; iy < met->ny; iy++)
03402             for (int ip = 0; ip < met->np; ip++) {
03403                 if (init)
03404                     dest[ix][iy][ip] = 0;
03405                 short aux = help[(ip * met->ny + iy) * met->nx + ix];
03406                 if ((fillval == 0 || aux != fillval)
03407                     && (missval == 0 || aux != missval)
03408                     && fabsf(aux * scalfac + offset) < 1e14f)
03409                     dest[ix][iy][ip] += scl * (aux * scalfac + offset);
03410                 else
03411                     dest[ix][iy][ip] = GSL_NAN;
03412             }
03413
03414     /* Free... */
03415     free(help);
03416 }
03417
03418 /* Unpacked data... */
03419 else {
03420
03421     /* Allocate... */
03422     float *help;
03423     ALLOC(help, float,
03424           EX * EY * EP);
03425
03426     /* Read fill value and missing value... */
03427     float fillval, missval;
03428     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03429         fillval = 0;
03430     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03431         missval = 0;
03432
03433     /* Write info... */
03434     LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
03435         varsel, fillval, missval);
03436
03437     /* Read data... */
03438     NC(nc_get_var_float(ncid, varid, help));
03439
03440     /* Copy and check data... */
03441 #pragma omp parallel for default(shared) num_threads(12)
03442     for (int ix = 0; ix < met->nx; ix++)
03443         for (int iy = 0; iy < met->ny; iy++)
03444             for (int ip = 0; ip < met->np; ip++) {
03445                 if (init)
03446                     dest[ix][iy][ip] = 0;
03447                 float aux = help[(ip * met->ny + iy) * met->nx + ix];
03448                 if ((fillval == 0 || aux != fillval)
03449                     && (missval == 0 || aux != missval)
03450                     && fabsf(aux) < 1e14f)
03451                     dest[ix][iy][ip] += scl * aux;
03452                 else
03453                     dest[ix][iy][ip] = GSL_NAN;
03454             }
03455
03456     /* Free... */
03457     free(help);
03458 }
03459
03460 /* Return... */
03461 return 1;
03462 }

```

5.19.2.30 read_met_help_2d() int read_met_help_2d (

```

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

```

Read and convert 2D variable from meteorological data file.

Definition at line 3466 of file libtrac.c.

```

03473     {
03474
03475     char varsel[LEN];
03476
03477     float offset, scalfac;
03478
03479     int varid;
03480
03481     /* Check if variable exists... */
03482     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03483         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03484             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03485             return 0;
03486         } else {
03487             sprintf(varsel, "%s", varname2);
03488         } else
03489             sprintf(varsel, "%s", varname);
03490
03491     /* Read packed data... */
03492     if (nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03493         && nc_get_att_float(ncid, varid, "scale_factor",
03494                             &scalfac) == NC_NOERR) {
03495
03496         /* Write info... */
03497         LOG(2, "Packed: scale_factor= %g / add_offset= %g", scalfac, offset);
03498
03499         /* Allocate... */
03500         short *help;
03501         ALLOC(help, short,
03502              EX * EY * EP);
03503
03504         /* Read fill value and missing value... */
03505         short fillval, missval;
03506         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03507             fillval = 0;
03508         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03509             missval = 0;
03510
03511         /* Write info... */
03512         LOG(2, "Read 2-D variable: %s"
03513              " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03514              varsel, fillval, missval, scalfac, offset);
03515
03516         /* Read data... */
03517         NC(nc_get_var_short(ncid, varid, help));
03518
03519         /* Copy and check data... */
03520         #pragma omp parallel for default(shared) num_threads(12)
03521         for (int ix = 0; ix < met->nx; ix++)
03522             for (int iy = 0; iy < met->ny; iy++) {
03523                 if (init)
03524                     dest[ix][iy] = 0;
03525                 short aux = help[iy * met->nx + ix];
03526                 if ((fillval == 0 || aux != fillval)
03527                     && (missval == 0 || aux != missval)
03528                     && fabsf(aux * scalfac + offset) < 1e14f)
03529                     dest[ix][iy] += scl * (aux * scalfac + offset);
03530                 else
03531                     dest[ix][iy] = GSL_NAN;
03532             }
03533
03534         /* Free... */
03535         free(help);
03536     }
03537
03538     /* Unpacked data... */
03539     else {
03540
03541         /* Allocate... */
03542         float *help;
03543         ALLOC(help, float,
03544              EX * EY);
03545
03546         /* Read fill value and missing value... */
03547         float fillval, missval;
03548         if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03549             fillval = 0;
03550         if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03551             missval = 0;
03552
03553         /* Write info... */
03554         LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03555              varsel, fillval, missval);

```

```

03556
03557     /* Read data... */
03558     NC(nc_get_var_float(ncid, varid, help));
03559
03560     /* Copy and check data... */
03561     #pragma omp parallel for default(shared) num_threads(12)
03562     for (int ix = 0; ix < met->nx; ix++)
03563         for (int iy = 0; iy < met->ny; iy++) {
03564             if (init)
03565                 dest[ix][iy] = 0;
03566             float aux = help[iy * met->nx + ix];
03567             if ((fillval == 0 || aux != fillval)
03568                 && (missval == 0 || aux != missval)
03569                 && fabsf(aux) < 1e14f)
03570                 dest[ix][iy] += scl * aux;
03571             else
03572                 dest[ix][iy] = GSL_NAN;
03573         }
03574
03575     /* Free... */
03576     free(help);
03577 }
03578
03579 /* Return... */
03580 return 1;
03581 }

```

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

Read meteorological data on vertical levels.

Definition at line 3585 of file libtrac.c.

```

03588     {
03589
03590     /* Set timer... */
03591     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03592     LOG(2, "Read level data...");
03593
03594     /* Read meteorological data... */
03595     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0, 1))
03596         ERRMSG("Cannot read temperature!");
03597     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0, 1))
03598         ERRMSG("Cannot read zonal wind!");
03599     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0, 1))
03600         ERRMSG("Cannot read meridional wind!");
03601     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f, 1))
03602         WARN("Cannot read vertical velocity!");
03603     if (!read_met_help_3d
03604         (ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O), 1))
03605         WARN("Cannot read specific humidity!");
03606     if (!read_met_help_3d
03607         (ncid, "o3", "O3", met, met->o3, (float) (MA / MO3), 1))
03608         WARN("Cannot read ozone data!");
03609     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03610         if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0, 1))
03611             WARN("Cannot read cloud liquid water content!");
03612         if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0, 1))
03613             WARN("Cannot read cloud ice water content!");
03614     }
03615     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03616         if (!read_met_help_3d
03617             (ncid, "crwc", "CRWC", met, met->lwc, 1.0, ctl->met_cloud == 2))
03618             WARN("Cannot read cloud rain water content!");
03619         if (!read_met_help_3d
03620             (ncid, "cswc", "CSWC", met, met->iwc, 1.0, ctl->met_cloud == 2))
03621             WARN("Cannot read cloud snow water content!");
03622     }
03623
03624     /* Transfer from model levels to pressure levels... */
03625     if (ctl->met_np > 0) {
03626
03627         /* Read pressure on model levels... */
03628         if (!read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f, 1))
03629             ERRMSG("Cannot read pressure on model levels!");
03630     }

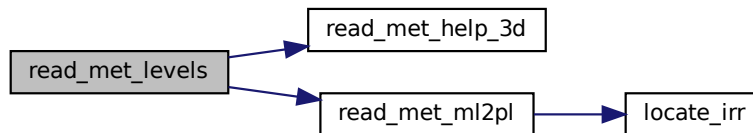
```

```

03631      /* Vertical interpolation from model to pressure levels... */
03632      read_met_ml2pl(ctl, met, met->t);
03633      read_met_ml2pl(ctl, met, met->u);
03634      read_met_ml2pl(ctl, met, met->v);
03635      read_met_ml2pl(ctl, met, met->w);
03636      read_met_ml2pl(ctl, met, met->h2o);
03637      read_met_ml2pl(ctl, met, met->o3);
03638      read_met_ml2pl(ctl, met, met->lwc);
03639      read_met_ml2pl(ctl, met, met->iwc);
03640
03641      /* Set new pressure levels... */
03642      met->np = ctl->met_np;
03643      for (int ip = 0; ip < met->np; ip++)
03644          met->p[ip] = ctl->met_p[ip];
03645  }
03646
03647      /* Check ordering of pressure levels... */
03648      for (int ip = 1; ip < met->np; ip++)
03649          if (met->p[ip - 1] < met->p[ip])
03650              ERRMSG("Pressure levels must be descending!");
03651  }

```

Here is the call graph for this function:



5.19.2.32 read_met_ml2pl() void read_met_ml2pl (

```

    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP] )

```

Convert meteorological data from model levels to pressure levels.

Definition at line 3655 of file libtrac.c.

```

03658      {
03659
03660          double aux[EP], p[EP];
03661
03662          /* Set timer... */
03663          SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03664          LOG(2, "Interpolate meteo data to pressure levels...");
03665
03666          /* Loop over columns... */
03667          #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03668          for (int ix = 0; ix < met->nx; ix++)
03669              for (int iy = 0; iy < met->ny; iy++) {
03670
03671                  /* Copy pressure profile... */
03672                  for (int ip = 0; ip < met->np; ip++)
03673                      p[ip] = met->p[ix][iy][ip];
03674
03675                  /* Interpolate... */
03676                  for (int ip = 0; ip < ctl->met_np; ip++) {
03677                      double pt = ctl->met_p[ip];
03678                      if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03679                          pt = p[0];
03680                      else if ((pt > p[met->np - 1] && p[1] > p[0])
03681                              || (pt < p[met->np - 1] && p[1] < p[0]))
03682                          pt = p[met->np - 1];
03683                      int ip2 = locate_irr(p, met->np, pt);

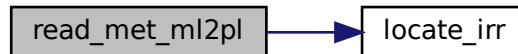
```

```

03684         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03685                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03686     }
03687
03688     /* Copy data... */
03689     for (int ip = 0; ip < ctl->met_np; ip++)
03690         var[ix][iy][ip] = (float) aux[ip];
03691 }
03692 }

```

Here is the call graph for this function:



5.19.2.33 read_met_pbl() void read_met_pbl (
 met_t * met)

Calculate pressure of the boundary layer.

Definition at line 3696 of file libtrac.c.

```

03697     {
03698
03699     /* Set timer... */
03700     SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
03701     LOG(2, "Calculate planetary boundary layer...");
03702
03703     /* Parameters used to estimate the height of the PBL
03704        (e.g., Vogelesang and Holtslag, 1996; Seidel et al., 2012)... */
03705     const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
03706
03707     /* Loop over grid points... */
03708     #pragma omp parallel for default(shared) collapse(2)
03709     for (int ix = 0; ix < met->nx; ix++)
03710         for (int iy = 0; iy < met->ny; iy++) {
03711
03712             /* Set bottom level of PBL... */
03713             double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
03714
03715             /* Find lowest level near the bottom... */
03716             int ip;
03717             for (ip = 1; ip < met->np; ip++)
03718                 if (met->p[ip] < pbl_bot)
03719                     break;
03720
03721             /* Get near surface data... */
03722             double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
03723                           met->p[ip], met->z[ix][iy][ip], pbl_bot);
03724             double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
03725                           met->p[ip], met->t[ix][iy][ip], pbl_bot);
03726             double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
03727                           met->p[ip], met->u[ix][iy][ip], pbl_bot);
03728             double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
03729                           met->p[ip], met->v[ix][iy][ip], pbl_bot);
03730             double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
03731                             met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
03732             double tvs = THETAVIRT(pbl_bot, ts, h2os);
03733
03734             /* Init... */
03735             double rib, rib_old = 0;
03736
03737             /* Loop over levels... */
03738             for (; ip < met->np; ip++) {
03739

```

```

03740      /* Get squared horizontal wind speed... */
03741      double vh2
03742      = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
03743      vh2 = GSL_MAX(vh2, SQR(umin));
03744
03745      /* Calculate bulk Richardson number... */
03746      rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
03747      * (THETAVIRT(met->p[ip], met->t[ix][iy][ip],
03748      met->h2o[ix][iy][ip]) - tvs) / vh2;
03749
03750      /* Check for critical value... */
03751      if (rib >= rib_crit) {
03752          met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
03753          rib, met->p[ip], rib_crit));
03754          if (met->pbl[ix][iy] > pbl_bot)
03755              met->pbl[ix][iy] = (float) pbl_bot;
03756          break;
03757      }
03758
03759      /* Save Richardson number... */
03760      rib_old = rib;
03761  }
03762  }
03763  }

```

5.19.2.34 read_met_periodic() void read_met_periodic (
met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 3767 of file libtrac.c.

```

03768      {
03769
03770      /* Set timer... */
03771      SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
03772      LOG(2, "Apply periodic boundary conditions...");
03773
03774      /* Check longitudes... */
03775      if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
03776      + met->lon[1] - met->lon[0] - 360) < 0.01))
03777          return;
03778
03779      /* Increase longitude counter... */
03780      if ((++met->nx) > EX)
03781          ERRMSG("Cannot create periodic boundary conditions!");
03782
03783      /* Set longitude... */
03784      met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
03785
03786      /* Loop over latitudes and pressure levels... */
03787      #pragma omp parallel for default(shared)
03788      for (int iy = 0; iy < met->ny; iy++) {
03789          met->ps[met->nx - 1][iy] = met->ps[0][iy];
03790          met->zs[met->nx - 1][iy] = met->zs[0][iy];
03791          met->ts[met->nx - 1][iy] = met->ts[0][iy];
03792          met->us[met->nx - 1][iy] = met->us[0][iy];
03793          met->vs[met->nx - 1][iy] = met->vs[0][iy];
03794          for (int ip = 0; ip < met->np; ip++) {
03795              met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
03796              met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
03797              met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
03798              met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
03799              met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
03800              met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
03801              met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
03802              met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
03803          }
03804      }
03805  }

```


5.19.2.35 read_met_pv() void read_met_pv (
 met_t * met)

Calculate potential vorticity.

Definition at line 3809 of file libtrac.c.

```

03810     {
03811
03812     double pows[EP];
03813
03814     /* Set timer... */
03815     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
03816     LOG(2, "Calculate potential vorticity...");
03817
03818     /* Set powers... */
03819     #pragma omp parallel for default(shared)
03820     for (int ip = 0; ip < met->np; ip++)
03821         pows[ip] = pow(1000. / met->p[ip], 0.286);
03822
03823     /* Loop over grid points... */
03824     #pragma omp parallel for default(shared)
03825     for (int ix = 0; ix < met->nx; ix++) {
03826
03827         /* Set indices... */
03828         int ix0 = GSL_MAX(ix - 1, 0);
03829         int ix1 = GSL_MIN(ix + 1, met->nx - 1);
03830
03831         /* Loop over grid points... */
03832         for (int iy = 0; iy < met->ny; iy++) {
03833
03834             /* Set indices... */
03835             int iy0 = GSL_MAX(iy - 1, 0);
03836             int iy1 = GSL_MIN(iy + 1, met->ny - 1);
03837
03838             /* Set auxiliary variables... */
03839             double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
03840             double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
03841             double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
03842             double c0 = cos(met->lat[iy0] / 180. * M_PI);
03843             double c1 = cos(met->lat[iy1] / 180. * M_PI);
03844             double cr = cos(latr / 180. * M_PI);
03845             double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
03846
03847             /* Loop over grid points... */
03848             for (int ip = 0; ip < met->np; ip++) {
03849
03850                 /* Get gradients in longitude... */
03851                 double dtdx
03852                     = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
03853                 double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
03854
03855                 /* Get gradients in latitude... */
03856                 double dtdy
03857                     = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03858                 double dudx
03859                     = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03860
03861                 /* Set indices... */
03862                 int ip0 = GSL_MAX(ip - 1, 0);
03863                 int ip1 = GSL_MIN(ip + 1, met->np - 1);
03864
03865                 /* Get gradients in pressure... */
03866                 double dtdp, dudp, dvdp;
03867                 double dp0 = 100. * (met->p[ip] - met->p[ip0]);
03868                 double dp1 = 100. * (met->p[ip1] - met->p[ip]);
03869                 if (ip != ip0 && ip != ip1) {
03870                     double denom = dp0 * dp1 * (dp0 + dp1);
03871                     dtdp = (dp0 * dp1 * met->t[ix][iy][ip1] * pows[ip1]
03872                         - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03873                         + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03874                         / denom;
03875                     dudp = (dp0 * dp1 * met->u[ix][iy][ip1]
03876                         - dp1 * dp1 * met->u[ix][iy][ip0]
03877                         + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03878                         / denom;
03879                     dvdp = (dp0 * dp1 * met->v[ix][iy][ip1]
03880                         - dp1 * dp1 * met->v[ix][iy][ip0]
03881                         + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03882                         / denom;
03883                 } else {
03884                     double denom = dp0 + dp1;
03885                     dtdp =
03886                         (met->t[ix][iy][ip1] * pows[ip1] -
03887                         met->t[ix][iy][ip0] * pows[ip0]) / denom;
03888                     dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;

```

```

03889         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03890     }
03891
03892     /* Calculate PV... */
03893     met->pv[ix][iy][ip] = (float)
03894         (1e6 * G0 *
03895          (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03896     }
03897 }
03898 }
03899
03900 /* Fix for polar regions... */
03901 #pragma omp parallel for default(shared)
03902 for (int ix = 0; ix < met->nx; ix++)
03903     for (int ip = 0; ip < met->np; ip++) {
03904         met->pv[ix][0][ip]
03905             = met->pv[ix][1][ip]
03906             = met->pv[ix][2][ip];
03907         met->pv[ix][met->ny - 1][ip]
03908             = met->pv[ix][met->ny - 2][ip]
03909             = met->pv[ix][met->ny - 3][ip];
03910     }
03911 }

```

5.19.2.36 read_met_sample() void read_met_sample (

```

    ctl_t * ctl,
    met_t * met )

```

Downsampling of meteorological data.

Definition at line 3915 of file libtrac.c.

```

03917     {
03918
03919         met_t *help;
03920
03921         /* Check parameters... */
03922         if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03923             && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03924             return;
03925
03926         /* Set timer... */
03927         SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
03928         LOG(2, "Downsampling of meteo data...");
03929
03930         /* Allocate... */
03931         ALLOC(help, met_t, 1);
03932
03933         /* Copy data... */
03934         help->nx = met->nx;
03935         help->ny = met->ny;
03936         help->np = met->np;
03937         memcpy(help->lon, met->lon, sizeof(met->lon));
03938         memcpy(help->lat, met->lat, sizeof(met->lat));
03939         memcpy(help->p, met->p, sizeof(met->p));
03940
03941         /* Smoothing... */
03942         for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
03943             for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
03944                 for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
03945                     help->ps[ix][iy] = 0;
03946                     help->zs[ix][iy] = 0;
03947                     help->ts[ix][iy] = 0;
03948                     help->us[ix][iy] = 0;
03949                     help->vs[ix][iy] = 0;
03950                     help->t[ix][iy][ip] = 0;
03951                     help->u[ix][iy][ip] = 0;
03952                     help->v[ix][iy][ip] = 0;
03953                     help->w[ix][iy][ip] = 0;
03954                     help->h2o[ix][iy][ip] = 0;
03955                     help->o3[ix][iy][ip] = 0;
03956                     help->lwc[ix][iy][ip] = 0;
03957                     help->iwc[ix][iy][ip] = 0;
03958                     float wsum = 0;
03959                     for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
03960                         ix2++) {
03961                         int ix3 = ix2;
03962                         if (ix3 < 0)
03963                             ix3 += met->nx;

```

```

03964         else if (ix3 >= met->nx)
03965             ix3 -= met->nx;
03966
03967         for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03968              iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03969             for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03970                  ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03971                 float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
03972                     * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
03973                     * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
03974                 help->ps[ix][iy] += w * met->ps[ix3][iy2];
03975                 help->zs[ix][iy] += w * met->zs[ix3][iy2];
03976                 help->ts[ix][iy] += w * met->ts[ix3][iy2];
03977                 help->us[ix][iy] += w * met->us[ix3][iy2];
03978                 help->vs[ix][iy] += w * met->vs[ix3][iy2];
03979                 help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03980                 help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03981                 help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03982                 help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03983                 help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03984                 help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03985                 help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03986                 help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03987                 wsum += w;
03988             }
03989         }
03990         help->ps[ix][iy] /= wsum;
03991         help->zs[ix][iy] /= wsum;
03992         help->ts[ix][iy] /= wsum;
03993         help->us[ix][iy] /= wsum;
03994         help->vs[ix][iy] /= wsum;
03995         help->t[ix][iy][ip] /= wsum;
03996         help->u[ix][iy][ip] /= wsum;
03997         help->v[ix][iy][ip] /= wsum;
03998         help->w[ix][iy][ip] /= wsum;
03999         help->h2o[ix][iy][ip] /= wsum;
04000         help->o3[ix][iy][ip] /= wsum;
04001         help->lwc[ix][iy][ip] /= wsum;
04002         help->iwc[ix][iy][ip] /= wsum;
04003     }
04004 }
04005 }
04006
04007 /* Downsampling... */
04008 met->nx = 0;
04009 for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04010     met->lon[met->nx] = help->lon[ix];
04011     met->ny = 0;
04012     for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04013         met->lat[met->ny] = help->lat[iy];
04014         met->ps[met->nx][met->ny] = help->ps[ix][iy];
04015         met->zs[met->nx][met->ny] = help->zs[ix][iy];
04016         met->ts[met->nx][met->ny] = help->ts[ix][iy];
04017         met->us[met->nx][met->ny] = help->us[ix][iy];
04018         met->vs[met->nx][met->ny] = help->vs[ix][iy];
04019         met->np = 0;
04020         for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04021             met->p[met->np] = help->p[ip];
04022             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04023             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04024             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04025             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04026             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04027             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04028             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
04029             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04030             met->np++;
04031         }
04032         met->ny++;
04033     }
04034     met->nx++;
04035 }
04036
04037 /* Free... */
04038 free(help);
04039 }

```

5.19.2.37 read_met_surface() void read_met_surface (

```

    int ncid,
    met_t * met )

```

Read surface data.

Definition at line 4043 of file libtrac.c.

```

04045     {
04046
04047     /* Set timer... */
04048     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04049     LOG(2, "Read surface data...");
04050
04051     /* Read surface pressure... */
04052     if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0f, 1)) {
04053         if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f, 1)) {
04054             WARN("Cannot not read surface pressure data (use lowest level!)");
04055             for (int ix = 0; ix < met->nx; ix++)
04056                 for (int iy = 0; iy < met->ny; iy++)
04057                     met->ps[ix][iy] = (float) met->p[0];
04058         }
04059     } else
04060         for (int ix = 0; ix < met->nx; ix++)
04061             for (int iy = 0; iy < met->ny; iy++)
04062                 met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04063
04064     /* Read geopotential height at the surface... */
04065     if (!read_met_help_2d
04066         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0)), 1))
04067         if (!read_met_help_2d
04068             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.), 1))
04069             WARN("Cannot read surface geopotential height!");
04070
04071     /* Read temperature at the surface... */
04072     if (!read_met_help_2d(ncid, "t2m", "T2M", met, met->ts, 1.0, 1))
04073         WARN("Cannot read surface temperature!");
04074
04075     /* Read zonal wind at the surface... */
04076     if (!read_met_help_2d(ncid, "u10m", "U10M", met, met->us, 1.0, 1))
04077         WARN("Cannot read surface zonal wind!");
04078
04079     /* Read meridional wind at the surface... */
04080     if (!read_met_help_2d(ncid, "v10m", "V10M", met, met->vs, 1.0, 1))
04081         WARN("Cannot read surface meridional wind!");
04082 }

```

Here is the call graph for this function:



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

Calculate tropopause data.

Definition at line 4086 of file libtrac.c.

```

04088     {
04089
04090     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04091         th2[200], z[EP], z2[200];
04092
04093     /* Set timer... */
04094     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04095     LOG(2, "Calculate tropopause...");
04096

```

```

04097  /* Get altitude and pressure profiles... */
04098  #pragma omp parallel for default(shared)
04099  for (int iz = 0; iz < met->np; iz++)
04100      z[iz] = Z(met->p[iz]);
04101  #pragma omp parallel for default(shared)
04102  for (int iz = 0; iz <= 190; iz++) {
04103      z2[iz] = 4.5 + 0.1 * iz;
04104      p2[iz] = P(z2[iz]);
04105  }
04106
04107  /* Do not calculate tropopause... */
04108  if (ctl->met_tropo == 0)
04109  #pragma omp parallel for default(shared) collapse(2)
04110  for (int ix = 0; ix < met->nx; ix++)
04111  for (int iy = 0; iy < met->ny; iy++)
04112      met->pt[ix][iy] = GSL_NAN;
04113
04114  /* Use tropopause climatology... */
04115  else if (ctl->met_tropo == 1) {
04116  #pragma omp parallel for default(shared) collapse(2)
04117  for (int ix = 0; ix < met->nx; ix++)
04118  for (int iy = 0; iy < met->ny; iy++)
04119      met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
04120  }
04121
04122  /* Use cold point... */
04123  else if (ctl->met_tropo == 2) {
04124
04125      /* Loop over grid points... */
04126  #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04127  for (int ix = 0; ix < met->nx; ix++)
04128  for (int iy = 0; iy < met->ny; iy++) {
04129
04130          /* Interpolate temperature profile... */
04131          for (int iz = 0; iz < met->np; iz++)
04132              t[iz] = met->t[ix][iy][iz];
04133          spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04134
04135          /* Find minimum... */
04136          int iz = (int) gsl_stats_min_index(t2, 1, 171);
04137          if (iz > 0 && iz < 170)
04138              met->pt[ix][iy] = (float) p2[iz];
04139          else
04140              met->pt[ix][iy] = GSL_NAN;
04141      }
04142  }
04143
04144  /* Use WMO definition... */
04145  else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04146
04147      /* Loop over grid points... */
04148  #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04149  for (int ix = 0; ix < met->nx; ix++)
04150  for (int iy = 0; iy < met->ny; iy++) {
04151
04152          /* Interpolate temperature profile... */
04153          int iz;
04154          for (iz = 0; iz < met->np; iz++)
04155              t[iz] = met->t[ix][iy][iz];
04156          spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04157
04158          /* Find 1st tropopause... */
04159          met->pt[ix][iy] = GSL_NAN;
04160          for (iz = 0; iz <= 170; iz++) {
04161              int found = 1;
04162              for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04163                  if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04164                      ctl->met_tropo_lapse) {
04165                      found = 0;
04166                      break;
04167                  }
04168              if (found) {
04169                  if (iz > 0 && iz < 170)
04170                      met->pt[ix][iy] = (float) p2[iz];
04171                  break;
04172              }
04173          }
04174
04175          /* Find 2nd tropopause... */
04176          if (ctl->met_tropo == 4) {
04177              met->pt[ix][iy] = GSL_NAN;
04178              for (; iz <= 170; iz++) {
04179                  int found = 1;
04180                  for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04181                      if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04182                          ctl->met_tropo_lapse_sep) {
04183                          found = 0;

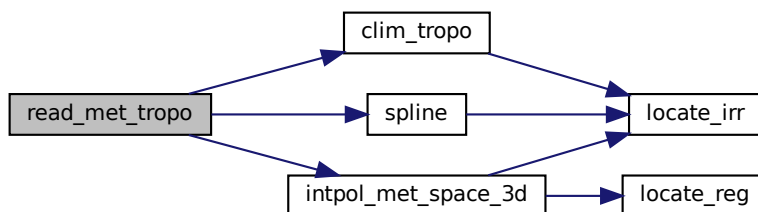
```

```

04184         break;
04185     }
04186     if (found)
04187         break;
04188 }
04189 for (; iz <= 170; iz++) {
04190     int found = 1;
04191     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04192         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04193             ctl->met_tropo_lapse) {
04194             found = 0;
04195             break;
04196         }
04197     if (found) {
04198         if (iz > 0 && iz < 170)
04199             met->pt[ix][iy] = (float) p2[iz];
04200         break;
04201     }
04202 }
04203 }
04204 }
04205 }
04206
04207 /* Use dynamical tropopause... */
04208 else if (ctl->met_tropo == 5) {
04209     /* Loop over grid points... */
04210 #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04211     for (int ix = 0; ix < met->nx; ix++)
04212         for (int iy = 0; iy < met->ny; iy++) {
04213             /* Interpolate potential vorticity profile... */
04214             for (int iz = 0; iz < met->np; iz++)
04215                 pv[iz] = met->pv[ix][iy][iz];
04216             spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04217
04218             /* Interpolate potential temperature profile... */
04219             for (int iz = 0; iz < met->np; iz++)
04220                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04221             spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04222
04223             /* Find dynamical tropopause... */
04224             met->pt[ix][iy] = GSL_NAN;
04225             for (int iz = 0; iz <= 170; iz++)
04226                 if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04227                     || th2[iz] >= ctl->met_tropo_theta) {
04228                     if (iz > 0 && iz < 170)
04229                         met->pt[ix][iy] = (float) p2[iz];
04230                     break;
04231                 }
04232         }
04233     }
04234 }
04235 }
04236
04237 else
04238     ERRMSG("Cannot calculate tropopause!");
04239
04240 /* Interpolate temperature, geopotential height, and water vapor vmr... */
04241 #pragma omp parallel for default(shared) collapse(2)
04242     for (int ix = 0; ix < met->nx; ix++)
04243         for (int iy = 0; iy < met->ny; iy++) {
04244             double h2ot, tt, zt;
04245             INTPOL_INIT;
04246             intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04247                               met->lat[iy], &tt, ci, cw, 1);
04248             intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04249                               met->lat[iy], &zt, ci, cw, 0);
04250             intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04251                               met->lat[iy], &h2ot, ci, cw, 0);
04252             met->tt[ix][iy] = (float) tt;
04253             met->zt[ix][iy] = (float) zt;
04254             met->h2ot[ix][iy] = (float) h2ot;
04255         }
04256 }

```

Here is the call graph for this function:



5.19.2.39 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 4260 of file [libtrac.c](#).

```

04267     {
04268
04269     FILE *in = NULL;
04270
04271     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
04272         rvarname[LEN], rval[LEN];
04273
04274     int contain = 0, i;
04275
04276     /* Open file... */
04277     if (filename[strlen(filename) - 1] != '-')
04278         if (!(in = fopen(filename, "r")))
04279             ERRMSG("Cannot open file!");
04280
04281     /* Set full variable name... */
04282     if (arridx >= 0) {
04283         sprintf(fullname1, "%s[%d]", varname, arridx);
04284         sprintf(fullname2, "%s[*]", varname);
04285     } else {
04286         sprintf(fullname1, "%s", varname);
04287         sprintf(fullname2, "%s", varname);
04288     }
04289
04290     /* Read data... */
04291     if (in != NULL)
04292         while (fgets(line, LEN, in))
04293             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
04294                 if (strcasecmp(rvarname, fullname1) == 0 ||
04295                     strcasecmp(rvarname, fullname2) == 0) {
04296                     contain = 1;
04297                     break;
04298                 }
04299     for (i = 1; i < argc - 1; i++)
04300         if (strcasecmp(argv[i], fullname1) == 0 ||
04301             strcasecmp(argv[i], fullname2) == 0) {
04302             sprintf(rval, "%s", argv[i + 1]);
04303             contain = 1;
04304             break;
04305         }
04306

```

```

04307  /* Close file... */
04308  if (in != NULL)
04309      fclose(in);
04310
04311  /* Check for missing variables... */
04312  if (!contain) {
04313      if (strlen(defvalue) > 0)
04314          sprintf(rval, "%s", defvalue);
04315      else
04316          ERRMSG("Missing variable %s!\n", fullname1);
04317  }
04318
04319  /* Write info... */
04320  LOG(1, "%s = %s", fullname1, rval);
04321
04322  /* Return values... */
04323  if (value != NULL)
04324      sprintf(value, "%s", rval);
04325  return atof(rval);
04326 }

```

5.19.2.40 sedi() double sedi (
 double *p*,
 double *T*,
 double *r_p*,
 double *rho_p*)

Calculate sedimentation velocity.

Definition at line 4330 of file [libtrac.c](#).

```

04334  {
04335
04336      double eta, G, K, lambda, rho, v;
04337
04338      /* Convert pressure from hPa to Pa... */
04339      p *= 100.;
04340
04341      /* Convert particle radius from microns to m... */
04342      r_p *= 1e-6;
04343
04344      /* Density of dry air [kg / m^3]... */
04345      rho = p / (RA * T);
04346
04347      /* Dynamic viscosity of air [kg / (m s)]... */
04348      eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04349
04350      /* Thermal velocity of an air molecule [m / s]... */
04351      v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04352
04353      /* Mean free path of an air molecule [m]... */
04354      lambda = 2. * eta / (rho * v);
04355
04356      /* Knudsen number for air (dimensionless)... */
04357      K = lambda / r_p;
04358
04359      /* Cunningham slip-flow correction (dimensionless)... */
04360      G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04361
04362      /* Sedimentation velocity [m / s]... */
04363      return 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
04364 }

```

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


```

    int n2,
    int method )

```

Spline interpolation.

Definition at line 4368 of file [libtrac.c](#).

```

04375     {
04376
04377     /* Cubic spline interpolation... */
04378     if (method == 1) {
04379
04380         /* Allocate... */
04381         gsl_interp_accel *acc;
04382         gsl_spline *s;
04383         acc = gsl_interp_accel_alloc();
04384         s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04385
04386         /* Interpolate profile... */
04387         gsl_spline_init(s, x, y, (size_t) n);
04388         for (int i = 0; i < n2; i++)
04389             if (x2[i] <= x[0])
04390                 y2[i] = y[0];
04391             else if (x2[i] >= x[n - 1])
04392                 y2[i] = y[n - 1];
04393             else
04394                 y2[i] = gsl_spline_eval(s, x2[i], acc);
04395
04396         /* Free... */
04397         gsl_spline_free(s);
04398         gsl_interp_accel_free(acc);
04399     }
04400
04401     /* Linear interpolation... */
04402     else {
04403         for (int i = 0; i < n2; i++)
04404             if (x2[i] <= x[0])
04405                 y2[i] = y[0];
04406             else if (x2[i] >= x[n - 1])
04407                 y2[i] = y[n - 1];
04408             else {
04409                 int idx = locate_irr(x, n, x2[i]);
04410                 y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04411             }
04412     }
04413 }

```

Here is the call graph for this function:



5.19.2.42 stddev() float stddev (
 float * data,
 int n)

Calculate standard deviation.

Definition at line 4417 of file [libtrac.c](#).

```

04419     {
04420
04421     if (n <= 0)

```

```

04422     return 0;
04423
04424     float mean = 0, var = 0;
04425
04426     for (int i = 0; i < n; ++i) {
04427         mean += data[i];
04428         var += SQR(data[i]);
04429     }
04430
04431     return sqrtf(var / (float) n - SQR(mean / (float) n));
04432 }

```

5.19.2.43 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 4436 of file libtrac.c.

```

04444     {
04445
04446     struct tm t0, t1;
04447
04448     t0.tm_year = 100;
04449     t0.tm_mon = 0;
04450     t0.tm_mday = 1;
04451     t0.tm_hour = 0;
04452     t0.tm_min = 0;
04453     t0.tm_sec = 0;
04454
04455     t1.tm_year = year - 1900;
04456     t1.tm_mon = mon - 1;
04457     t1.tm_mday = day;
04458     t1.tm_hour = hour;
04459     t1.tm_min = min;
04460     t1.tm_sec = sec;
04461
04462     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04463 }

```

5.19.2.44 timer() void timer (

```

    const char * name,
    const char * group,
    int output )

```

Measure wall-clock time.

Definition at line 4467 of file libtrac.c.

```

04470     {
04471
04472     static char names[NTIMER][100], groups[NTIMER][100];
04473
04474     static double rt_name[NTIMER], rt_group[NTIMER], t0, t1;
04475
04476     static int iname = -1, igroup = -1, nname, ngroup;
04477
04478     /* Get time... */
04479     t1 = omp_get_wtime();
04480
04481     /* Add elapsed time to current timers... */

```

```

04482     if (iname >= 0)
04483         rt_name[iname] += t1 - t0;
04484     if (igroup >= 0)
04485         rt_group[igroup] += t1 - t0;
04486
04487     /* Report timers... */
04488     if (output) {
04489         for (int i = 0; i < nname; i++)
04490             LOG(1, "TIMER_%s = %.3f s", names[i], rt_name[i]);
04491         for (int i = 0; i < ngroup; i++)
04492             LOG(1, "TIMER_%s = %.3f s", groups[i], rt_group[i]);
04493         double total = 0.0;
04494         for (int i = 0; i < nname; i++)
04495             total += rt_name[i];
04496         LOG(1, "TIMER_TOTAL = %.3f s", total);
04497     }
04498
04499     /* Identify IDs of next timer... */
04500     for (iname = 0; iname < nname; iname++)
04501         if (strcasecmp(name, names[iname]) == 0)
04502             break;
04503     for (igroup = 0; igroup < ngroup; igroup++)
04504         if (strcasecmp(group, groups[igroup]) == 0)
04505             break;
04506
04507     /* Check whether this is a new timer... */
04508     if (iname >= nname) {
04509         sprintf(names[iname], "%s", name);
04510         if ((++nname) > NTIMER)
04511             ERRMSG("Too many timers!");
04512     }
04513
04514     /* Check whether this is a new group... */
04515     if (igroup >= ngroup) {
04516         sprintf(groups[igroup], "%s", group);
04517         if ((++ngroup) > NTIMER)
04518             ERRMSG("Too many groups!");
04519     }
04520
04521     /* Save starting time... */
04522     t0 = t1;
04523 }

```

5.19.2.45 tropo_weight() double tropo_weight (
 double t,
 double lat,
 double p)

Get weighting factor based on tropopause distance.

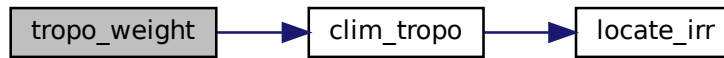
Definition at line 4527 of file libtrac.c.

```

04530     {
04531
04532     /* Get tropopause pressure... */
04533     double pt = clim_tropo(t, lat);
04534
04535     /* Get pressure range... */
04536     double p1 = pt * 0.866877899;
04537     double p0 = pt / 0.866877899;
04538
04539     /* Get weighting factor... */
04540     if (p > p0)
04541         return 1;
04542     else if (p < p1)
04543         return 0;
04544     else
04545         return LIN(p0, 1.0, p1, 0.0, p);
04546 }

```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 4550 of file libtrac.c.

```

04554     {
04555
04556     FILE *in, *out;
04557
04558     char line[LEN];
04559
04560     double r, t0, t1;
04561
04562     int year, mon, day, hour, min, sec;
04563
04564     /* Set timer... */
04565     SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
04566
04567     /* Set time interval for output... */
04568     t0 = t - 0.5 * ctl->dt_mod;
04569     t1 = t + 0.5 * ctl->dt_mod;
04570
04571     /* Write info... */
04572     LOG(1, "Write atmospheric data: %s", filename);
04573
04574     /* Write ASCII data... */
04575     if (ctl->atm_type == 0) {
04576
04577         /* Check if gnuplot output is requested... */
04578         if (ctl->atm_gpfile[0] != '-') {
04579
04580             /* Create gnuplot pipe... */
04581             if (!(out = popen("gnuplot", "w")))
04582                 ERRMSG("Cannot create pipe to gnuplot!");
04583
04584             /* Set plot filename... */
04585             fprintf(out, "set out \"%s.png\"\n", filename);
04586
04587             /* Set time string... */
04588             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04589             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04590                 year, mon, day, hour, min);
04591
04592             /* Dump gnuplot file to pipe... */
04593             if (!(in = fopen(ctl->atm_gpfile, "r")))
04594                 ERRMSG("Cannot open file!");
04595             while (fgets(line, LEN, in))
04596                 fprintf(out, "%s", line);
04597             fclose(in);
04598         }
04599
04600     else {
04601
04602         /* Create file... */
04603         if (!(out = fopen(filename, "w")))
  
```

```

04604         ERRMSG("Cannot create file!");
04605     }
04606
04607     /* Write header... */
04608     fprintf(out,
04609         "# $1 = time [s]\n"
04610         "# $2 = altitude [km]\n"
04611         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04612     for (int iq = 0; iq < ctl->nq; iq++)
04613         fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
04614             ctl->qnt_unit[iq]);
04615     fprintf(out, "\n");
04616
04617     /* Write data... */
04618     for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
04619
04620         /* Check time... */
04621         if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
04622             continue;
04623
04624         /* Write output... */
04625         fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
04626             atm->lon[ip], atm->lat[ip]);
04627         for (int iq = 0; iq < ctl->nq; iq++) {
04628             fprintf(out, " ");
04629             if (ctl->atm_filter == 1
04630                 && (atm->time[ip] < t0 || atm->time[ip] > t1))
04631                 fprintf(out, ctl->qnt_format[iq], GSL_NAN);
04632             else
04633                 fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04634         }
04635         fprintf(out, "\n");
04636     }
04637
04638     /* Close file... */
04639     fclose(out);
04640 }
04641
04642 /* Write binary data... */
04643 else if (ctl->atm_type == 1) {
04644
04645     /* Create file... */
04646     if (!(out = fopen(filename, "w")))
04647         ERRMSG("Cannot create file!");
04648
04649     /* Write data... */
04650     FWRITE(&atm->np, int,
04651         1,
04652         out);
04653     FWRITE(atm->time, double,
04654         (size_t) atm->np,
04655         out);
04656     FWRITE(atm->p, double,
04657         (size_t) atm->np,
04658         out);
04659     FWRITE(atm->lon, double,
04660         (size_t) atm->np,
04661         out);
04662     FWRITE(atm->lat, double,
04663         (size_t) atm->np,
04664         out);
04665     for (int iq = 0; iq < ctl->nq; iq++)
04666         FWRITE(atm->q[iq], double,
04667             (size_t) atm->np,
04668             out);
04669
04670     /* Close file... */
04671     fclose(out);
04672 }
04673
04674 /* Error... */
04675 else
04676     ERRMSG("Atmospheric data type not supported!");
04677
04678 /* Write info... */
04679 double mini, maxi;
04680 LOG(2, "Number of particles: %d", atm->np);
04681 gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
04682 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04683 gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
04684 LOG(2, "Altitude range: %g ... %g km", Z(mini), Z(maxi));
04685 LOG(2, "Pressure range: %g ... %g hPa", mini, maxi);
04686 gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
04687 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04688 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
04689 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04690 for (int iq = 0; iq < ctl->nq; iq++) {

```

```

04691     char msg[LEN];
04692     sprintf(msg, "Quantity %s range: %s ... %s %s",
04693             ctl->qnt_name[iq], ctl->qnt_format[iq],
04694             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
04695     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
04696     LOG(2, msg, mini, maxi);
04697 }
04698 }

```

Here is the call graph for this function:



5.19.2.47 write_csi() void write_csi (

```

    const char * filename,
    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write CSI data.

Definition at line 4702 of file libtrac.c.

```

04706     {
04707
04708     static FILE *in, *out;
04709
04710     static char line[LEN];
04711
04712     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ], rt, rt_old,
04713             rz, rlon, rlat, robs, t0, t1, area[GY], dlon, dlat, dz, lat,
04714             x[1000000], y[1000000], work[2000000];
04715
04716     static int obscount[GX][GY][GZ], ct, cx, cy, cz, ip, ix, iy, iz, n;
04717
04718     /* Set timer... */
04719     SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
04720
04721     /* Init... */
04722     if (t == ctl->t_start) {
04723
04724         /* Check quantity index for mass... */
04725         if (ctl->qnt_m < 0)
04726             ERRMSG("Need quantity mass!");
04727
04728         /* Open observation data file... */
04729         LOG(1, "Read CSI observation data: %s", ctl->csi_obsfile);
04730         if (!(in = fopen(ctl->csi_obsfile, "r")))
04731             ERRMSG("Cannot open file!");
04732
04733         /* Initialize time for file input... */
04734         rt_old = -1e99;
04735
04736         /* Create new file... */
04737         LOG(1, "Write CSI data: %s", filename);
04738         if (!(out = fopen(filename, "w")))
04739             ERRMSG("Cannot create file!");
04740
04741         /* Write header... */
04742         fprintf(out,
04743             "# $1 = time [s]\n"
04744             "# $2 = number of hits (cx)\n"
04745             "# $3 = number of misses (cy)\n"

```

```

04746         "# $4 = number of false alarms (cz)\n"
04747         "# $5 = number of observations (cx + cy)\n"
04748         "# $6 = number of forecasts (cx + cz)\n"
04749         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
04750         "# $8 = probability of detection (POD) [%%]\n"
04751         "# $9 = false alarm rate (FAR) [%%]\n"
04752         "# $10 = critical success index (CSI) [%%]\n");
04753     fprintf(out,
04754         "# $11 = hits associated with random chance\n"
04755         "# $12 = equitable threat score (ETS) [%%]\n"
04756         "# $13 = Pearson linear correlation coefficient\n"
04757         "# $14 = Spearman rank-order correlation coefficient\n"
04758         "# $15 = column density mean error (F - O) [kg/m^2]\n"
04759         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
04760         "# $17 = column density mean absolute error [kg/m^2]\n"
04761         "# $18 = number of data points\n\n");
04762
04763     /* Set grid box size... */
04764     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
04765     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
04766     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
04767
04768     /* Set horizontal coordinates... */
04769     for (iy = 0; iy < ctl->csi_ny; iy++) {
04770         lat = ctl->csi_lat0 + dlat * (iy + 0.5);
04771         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
04772     }
04773 }
04774
04775 /* Set time interval... */
04776 t0 = t - 0.5 * ctl->dt_mod;
04777 t1 = t + 0.5 * ctl->dt_mod;
04778
04779 /* Initialize grid cells... */
04780 #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(3)
04781 for (ix = 0; ix < ctl->csi_nx; ix++)
04782     for (iy = 0; iy < ctl->csi_ny; iy++)
04783         for (iz = 0; iz < ctl->csi_nz; iz++)
04784             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
04785
04786 /* Read observation data... */
04787 while (fgets(line, LEN, in)) {
04788
04789     /* Read data... */
04790     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
04791         5)
04792         continue;
04793
04794     /* Check time... */
04795     if (rt < t0)
04796         continue;
04797     if (rt > t1)
04798         break;
04799     if (rt < rt_old)
04800         ERRMSG("Time must be ascending!");
04801     rt_old = rt;
04802
04803     /* Check observation data... */
04804     if (!isfinite(robs))
04805         continue;
04806
04807     /* Calculate indices... */
04808     ix = (int) ((rlon - ctl->csi_lon0) / dlon);
04809     iy = (int) ((rlat - ctl->csi_lat0) / dlat);
04810     iz = (int) ((rz - ctl->csi_z0) / dz);
04811
04812     /* Check indices... */
04813     if (ix < 0 || ix >= ctl->csi_nx ||
04814         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04815         continue;
04816
04817     /* Get mean observation index... */
04818     obsmean[ix][iy][iz] += robs;
04819     obscount[ix][iy][iz]++;
04820 }
04821
04822 /* Analyze model data... */
04823 for (ip = 0; ip < atm->np; ip++) {
04824
04825     /* Check time... */
04826     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04827         continue;
04828
04829     /* Get indices... */
04830     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
04831     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
04832     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);

```

```

04833
04834 /* Check indices... */
04835 if (ix < 0 || ix >= ctl->csi_nx ||
04836     iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04837     continue;
04838
04839 /* Get total mass in grid cell... */
04840 modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04841 }
04842
04843 /* Analyze all grid cells... */
04844 for (ix = 0; ix < ctl->csi_nx; ix++)
04845     for (iy = 0; iy < ctl->csi_ny; iy++)
04846         for (iz = 0; iz < ctl->csi_nz; iz++) {
04847
04848             /* Calculate mean observation index... */
04849             if (obscount[ix][iy][iz] > 0)
04850                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
04851
04852             /* Calculate column density... */
04853             if (modmean[ix][iy][iz] > 0)
04854                 modmean[ix][iy][iz] /= (1e6 * area[iy]);
04855
04856             /* Calculate CSI... */
04857             if (obscount[ix][iy][iz] > 0) {
04858                 ct++;
04859                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04860                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04861                     cx++;
04862                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04863                     modmean[ix][iy][iz] < ctl->csi_modmin)
04864                     cy++;
04865                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
04866                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04867                     cz++;
04868             }
04869
04870             /* Save data for other verification statistics... */
04871             if (obscount[ix][iy][iz] > 0
04872                 && (obsmean[ix][iy][iz] >= ctl->csi_obsmin
04873                     || modmean[ix][iy][iz] >= ctl->csi_modmin)) {
04874                 x[n] = modmean[ix][iy][iz];
04875                 y[n] = obsmean[ix][iy][iz];
04876                 if ((++n) > 1000000)
04877                     ERRMSG("Too many data points to calculate statistics!");
04878             }
04879         }
04880
04881 /* Write output... */
04882 if (fmod(t, ctl->csi_dt_out) == 0) {
04883
04884     /* Calculate verification statistics
04885        (https://www.cawcr.gov.au/projects/verification/) ... */
04886     int nobs = cx + cy;
04887     int nfor = cx + cz;
04888     double bias = (nobs > 0) ? 100. * nfor / nobs : GSL_NAN;
04889     double pod = (nobs > 0) ? (100. * cx) / nobs : GSL_NAN;
04890     double far = (nfor > 0) ? (100. * cz) / nfor : GSL_NAN;
04891     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
04892     double cx_rd = (ct > 0) ? (1. * nobs * nfor) / ct : GSL_NAN;
04893     double ets = (cx + cy + cz - cx_rd > 0) ?
04894         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
04895     double rho_p =
04896         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
04897     double rho_s =
04898         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
04899     for (int i = 0; i < n; i++)
04900         work[i] = x[i] - y[i];
04901     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
04902     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
04903         0.0) : GSL_NAN;
04904     double absdev =
04905         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
04906
04907     /* Write... */
04908     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %d\n",
04909         t, cx, cy, cz, nobs, nfor, bias, pod, far, csi, cx_rd, ets,
04910         rho_p, rho_s, mean, rmse, absdev, n);
04911
04912     /* Set counters to zero... */
04913     n = ct = cx = cy = cz = 0;
04914 }
04915
04916 /* Close file... */
04917 if (t == ctl->t_stop)
04918     fclose(out);
04919 }

```


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

Write ensemble data.

Definition at line 4923 of file libtrac.c.

```

04927     {
04928
04929         static FILE *out;
04930
04931         static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
04932             t0, t1, x[NENS][3], xm[3];
04933
04934         static int ip, iq;
04935
04936         static size_t i, n;
04937
04938         /* Set timer... */
04939         SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
04940
04941         /* Init... */
04942         if (t == ctl->t_start) {
04943
04944             /* Check quantities... */
04945             if (ctl->qnt_ens < 0)
04946                 ERRMSG("Missing ensemble IDs!");
04947
04948             /* Create new file... */
04949             LOG(1, "Write ensemble data: %s", filename);
04950             if (!(out = fopen(filename, "w")))
04951                 ERRMSG("Cannot create file!");
04952
04953             /* Write header... */
04954             fprintf(out,
04955                 "# $1 = time [s]\n"
04956                 "# $2 = altitude [km]\n"
04957                 "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04958             for (iq = 0; iq < ctl->nq; iq++)
04959                 fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
04960                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04961             for (iq = 0; iq < ctl->nq; iq++)
04962                 fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
04963                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04964             fprintf(out, "# $d = number of members\n", 5 + 2 * ctl->nq);
04965         }
04966
04967         /* Set time interval... */
04968         t0 = t - 0.5 * ctl->dt_mod;
04969         t1 = t + 0.5 * ctl->dt_mod;
04970
04971         /* Init... */
04972         ens = GSL_NAN;
04973         n = 0;
04974
04975         /* Loop over air parcels... */
04976         for (ip = 0; ip < atm->np; ip++) {
04977
04978             /* Check time... */
04979             if (atm->time[ip] < t0 || atm->time[ip] > t1)
04980                 continue;
04981
04982             /* Check ensemble id... */
04983             if (atm->q[ctl->qnt_ens][ip] != ens) {
04984
04985                 /* Write results... */
04986                 if (n > 0) {
04987
04988                     /* Get mean position... */
04989                     xm[0] = xm[1] = xm[2] = 0;
04990                     for (i = 0; i < n; i++) {
04991                         xm[0] += x[i][0] / (double) n;
04992                         xm[1] += x[i][1] / (double) n;
04993                         xm[2] += x[i][2] / (double) n;
04994                     }
04995                     cart2geo(xm, &dummy, &lon, &lat);

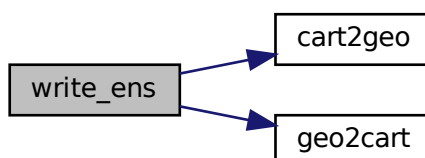
```

```

04996         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
04997             lat);
04998
04999         /* Get quantity statistics... */
05000         for (iq = 0; iq < ctl->nq; iq++) {
05001             fprintf(out, " ");
05002             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
05003         }
05004         for (iq = 0; iq < ctl->nq; iq++) {
05005             fprintf(out, " ");
05006             fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
05007         }
05008         fprintf(out, " %zu\n", n);
05009     }
05010
05011     /* Init new ensemble... */
05012     ens = atm->q[ctl->qnt_ens][ip];
05013     n = 0;
05014 }
05015
05016 /* Save data... */
05017 p[n] = atm->p[ip];
05018 geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
05019 for (iq = 0; iq < ctl->nq; iq++)
05020     q[iq][n] = atm->q[iq][ip];
05021 if ((++n) >= NENS)
05022     ERRMSG("Too many data points!");
05023 }
05024
05025 /* Write results... */
05026 if (n > 0) {
05027
05028     /* Get mean position... */
05029     xm[0] = xm[1] = xm[2] = 0;
05030     for (i = 0; i < n; i++) {
05031         xm[0] += x[i][0] / (double) n;
05032         xm[1] += x[i][1] / (double) n;
05033         xm[2] += x[i][2] / (double) n;
05034     }
05035     cart2geo(xm, &dummy, &lon, &lat);
05036     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
05037
05038     /* Get quantity statistics... */
05039     for (iq = 0; iq < ctl->nq; iq++) {
05040         fprintf(out, " ");
05041         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
05042     }
05043     for (iq = 0; iq < ctl->nq; iq++) {
05044         fprintf(out, " ");
05045         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
05046     }
05047     fprintf(out, " %zu\n", n);
05048 }
05049
05050 /* Close file... */
05051 if (t == ctl->t_stop)
05052     fclose(out);
05053 }

```

Here is the call graph for this function:



```

5.19.2.49 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 5057 of file libtrac.c.

```

50563     {
50564
50565     FILE *in, *out;
50566
50567     char line[LEN];
50568
50569     static double mass[GX][GY][GZ], vmr[GX][GY][GZ], vmr_expl, vmr_impl,
50570     z[GZ], dz, lon[GX], dlon, lat[GY], dlat, area[GY], rho_air,
50571     press[GZ], temp, cd, t0, t1, r;
50572
50573     static int ip, ix, *ixs, iy, *iys, iz, *izs, np[GX][GY][GZ], year, mon, day,
50574     hour, min, sec;
50575
50576     /* Set timer... */
50577     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
50578
50579     /* Check dimensions... */
50580     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
50581         ERRMSG("Grid dimensions too large!");
50582
50583     /* Set grid box size... */
50584     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
50585     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
50586     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
50587
50588     /* Set vertical coordinates... */
50589     #pragma omp parallel for default(shared) private(iz)
50590     for (iz = 0; iz < ctl->grid_nz; iz++) {
50591         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
50592         press[iz] = P(z[iz]);
50593     }
50594
50595     /* Set horizontal coordinates... */
50596     for (ix = 0; ix < ctl->grid_nx; ix++)
50597         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
50598     #pragma omp parallel for default(shared) private(iy)
50599     for (iy = 0; iy < ctl->grid_ny; iy++) {
50600         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
50601         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
50602             * cos(lat[iy] * M_PI / 180.);
50603     }
50604
50605     /* Set time interval for output... */
50606     t0 = t - 0.5 * ctl->dt_mod;
50607     t1 = t + 0.5 * ctl->dt_mod;
50608
50609     /* Initialize grid... */
50610     #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(3)
50611     for (ix = 0; ix < ctl->grid_nx; ix++)
50612         for (iy = 0; iy < ctl->grid_ny; iy++)
50613             for (iz = 0; iz < ctl->grid_nz; iz++) {
50614                 mass[ix][iy][iz] = 0;
50615                 vmr[ix][iy][iz] = 0;
50616                 np[ix][iy][iz] = 0;
50617             }
50618
50619     /* Allocate... */
50620     ALLOC(ixs, int,
50621         atm->np);
50622     ALLOC(iys, int,
50623         atm->np);
50624     ALLOC(izs, int,
50625         atm->np);
50626
50627     /* Get indices... */
50628     #pragma omp parallel for default(shared) private(ip)
50629     for (ip = 0; ip < atm->np; ip++) {
50630         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
50631         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
50632         izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
50633         if (atm->time[ip] < t0 || atm->time[ip] > t1

```

```

05134         || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05135         || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05136         || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05137     izs[ip] = -1;
05138 }
05139
05140 /* Average data... */
05141 for (ip = 0; ip < atm->np; ip++)
05142     if (izs[ip] >= 0) {
05143         np[ixs[ip]][iys[ip]][izs[ip]]++;
05144         if (ctl->qnt_m >= 0)
05145             mass[ixs[ip]][iys[ip]][izs[ip]] += atm->q[ctl->qnt_m][ip];
05146         if (ctl->qnt_vmr >= 0)
05147             vmr[ixs[ip]][iys[ip]][izs[ip]] += atm->q[ctl->qnt_vmr][ip];
05148     }
05149
05150 /* Free... */
05151 free(ixs);
05152 free(iys);
05153 free(izs);
05154
05155 /* Check if gnuplot output is requested... */
05156 if (ctl->grid_gpfile[0] != '-') {
05157
05158     /* Write info... */
05159     LOG(1, "Plot grid data: %s.png", filename);
05160
05161     /* Create gnuplot pipe... */
05162     if (!(out = popen("gnuplot", "w")))
05163         ERRMSG("Cannot create pipe to gnuplot!");
05164
05165     /* Set plot filename... */
05166     fprintf(out, "set out \"%s.png\"\n", filename);
05167
05168     /* Set time string... */
05169     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05170     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05171             year, mon, day, hour, min);
05172
05173     /* Dump gnuplot file to pipe... */
05174     if (!(in = fopen(ctl->grid_gpfile, "r")))
05175         ERRMSG("Cannot open file!");
05176     while (fgets(line, LEN, in))
05177         fprintf(out, "%s", line);
05178     fclose(in);
05179 }
05180
05181 else {
05182
05183     /* Write info... */
05184     LOG(1, "Write grid data: %s", filename);
05185
05186     /* Create file... */
05187     if (!(out = fopen(filename, "w")))
05188         ERRMSG("Cannot create file!");
05189 }
05190
05191 /* Write header... */
05192 fprintf(out,
05193         "# $1 = time [s]\n"
05194         "# $2 = altitude [km]\n"
05195         "# $3 = longitude [deg]\n"
05196         "# $4 = latitude [deg]\n"
05197         "# $5 = surface area [km^2]\n"
05198         "# $6 = layer width [km]\n"
05199         "# $7 = number of particles [l]\n"
05200         "# $8 = column density (implicit) [kg/m^2]\n"
05201         "# $9 = volume mixing ratio (implicit) [ppv]\n"
05202         "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
05203
05204 /* Write data... */
05205 for (ix = 0; ix < ctl->grid_nx; ix++) {
05206     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
05207         fprintf(out, "\n");
05208     for (iy = 0; iy < ctl->grid_ny; iy++) {
05209         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
05210             fprintf(out, "\n");
05211         for (iz = 0; iz < ctl->grid_nz; iz++)
05212             if (!ctl->grid_sparse || mass[ix][iy][iz] > 0 || vmr[ix][iy][iz] > 0) {
05213
05214                 /* Calculate column density... */
05215                 if (ctl->qnt_m >= 0)
05216                     cd = mass[ix][iy][iz] / (1e6 * area[iy]);
05217                 else
05218                     cd = GSL_NAN;
05219
05220                 /* Calculate volume mixing ratio (implicit)... */

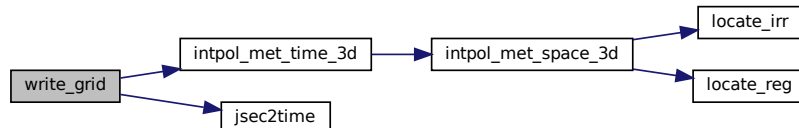
```

```

05221     if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05222         vmr_impl = 0;
05223         if (mass[ix][iy][iz] > 0) {
05224             /* Get temperature... */
05225             INTPOL_INIT;
05226             intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05227                             lon[ix], lat[iy], &temp, ci, cw, 1);
05228
05229             /* Calculate density of air... */
05230             rho_air = 100. * press[iz] / (RA * temp);
05231
05232             /* Calculate volume mixing ratio... */
05233             vmr_impl = MA / ctl->molmass * mass[ix][iy][iz]
05234                 / (rho_air * 1e6 * area[iy] * 1e3 * dz);
05235         }
05236     } else
05237     {
05238         vmr_impl = GSL_NAN;
05239
05240         /* Calculate volume mixing ratio (explicit)... */
05241         if (ctl->qnt_vmr >= 0 && np[ix][iy][iz] > 0)
05242             vmr_expl = vmr[ix][iy][iz] / np[ix][iy][iz];
05243         else
05244             vmr_expl = GSL_NAN;
05245
05246         /* Write output... */
05247         fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", t, z[iz],
05248                 lon[ix], lat[iy], area[iy], dz, np[ix][iy][iz], cd,
05249                 vmr_impl, vmr_expl);
05250     }
05251 }
05252 }
05253
05254 /* Close file... */
05255 fclose(out);
05256 }

```

Here is the call graph for this function:



5.19.2.50 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 5260 of file libtrac.c.

```

05266     {
05267
05268         static FILE *in, *out;
05269
05270         static char line[LEN];
05271
05272         static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rt_old,
05273             rz, rlon, rlat, robs, t0, t1, area[GY], dz, dlon, dlat,
05274             lon[GX], lat[GY], z[GZ], press[GZ], temp, rho_air, vmr, h2o, o3;

```

```

05275
05276 static int obscount[GX][GY], ip, ix, iy, iz, okay;
05277
05278 /* Set timer... */
05279 SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
05280
05281 /* Init... */
05282 if (t == ctl->t_start) {
05283
05284     /* Check quantity index for mass... */
05285     if (ctl->qnt_m < 0)
05286         ERRMSG("Need quantity mass!");
05287
05288     /* Check dimensions... */
05289     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
05290         ERRMSG("Grid dimensions too large!");
05291
05292     /* Check molar mass... */
05293     if (ctl->molmass <= 0)
05294         ERRMSG("Specify molar mass!");
05295
05296     /* Open observation data file... */
05297     LOG(1, "Read profile observation data: %s", ctl->prof_obsfile);
05298     if (! (in = fopen(ctl->prof_obsfile, "r")))
05299         ERRMSG("Cannot open file!");
05300
05301     /* Initialize time for file input... */
05302     rt_old = -1e99;
05303
05304     /* Create new output file... */
05305     LOG(1, "Write profile data: %s", filename);
05306     if (! (out = fopen(filename, "w")))
05307         ERRMSG("Cannot create file!");
05308
05309     /* Write header... */
05310     fprintf(out,
05311         "# $1 = time [s]\n"
05312         "# $2 = altitude [km]\n"
05313         "# $3 = longitude [deg]\n"
05314         "# $4 = latitude [deg]\n"
05315         "# $5 = pressure [hPa]\n"
05316         "# $6 = temperature [K]\n"
05317         "# $7 = volume mixing ratio [ppv]\n"
05318         "# $8 = H2O volume mixing ratio [ppv]\n"
05319         "# $9 = O3 volume mixing ratio [ppv]\n"
05320         "# $10 = observed BT index [K]\n"
05321         "# $11 = number of observations\n");
05322
05323     /* Set grid box size... */
05324     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
05325     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
05326     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
05327
05328     /* Set vertical coordinates... */
05329     for (iz = 0; iz < ctl->prof_nz; iz++) {
05330         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
05331         press[iz] = P(z[iz]);
05332     }
05333
05334     /* Set horizontal coordinates... */
05335     for (ix = 0; ix < ctl->prof_nx; ix++)
05336         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
05337     for (iy = 0; iy < ctl->prof_ny; iy++) {
05338         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
05339         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05340             * cos(lat[iy] * M_PI / 180.);
05341     }
05342 }
05343
05344 /* Set time interval... */
05345 t0 = t - 0.5 * ctl->dt_mod;
05346 t1 = t + 0.5 * ctl->dt_mod;
05347
05348 /* Initialize... */
05349 #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(2)
05350 for (ix = 0; ix < ctl->prof_nx; ix++)
05351     for (iy = 0; iy < ctl->prof_ny; iy++) {
05352         obsmean[ix][iy] = 0;
05353         obscount[ix][iy] = 0;
05354         for (iz = 0; iz < ctl->prof_nz; iz++)
05355             mass[ix][iy][iz] = 0;
05356     }
05357
05358 /* Read observation data... */
05359 while (fgets(line, LEN, in)) {
05360
05361     /* Read data... */

```

```

05362     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
05363         5)
05364         continue;
05365
05366     /* Check time... */
05367     if (rt < t0)
05368         continue;
05369     if (rt > t1)
05370         break;
05371     if (rt < rt_old)
05372         ERRMSG("Time must be ascending!");
05373     rt_old = rt;
05374
05375     /* Check observation data... */
05376     if (!isfinite(robs))
05377         continue;
05378
05379     /* Calculate indices... */
05380     ix = (int) ((rln - ctl->prof_lon0) / dlon);
05381     iy = (int) ((rln - ctl->prof_lat0) / dlat);
05382
05383     /* Check indices... */
05384     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
05385         continue;
05386
05387     /* Get mean observation index... */
05388     obsmean[ix][iy] += robs;
05389
05390     /* Count observations... */
05391     obscount[ix][iy]++;
05392 }
05393
05394 /* Analyze model data... */
05395 for (ip = 0; ip < atm->np; ip++) {
05396
05397     /* Check time... */
05398     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05399         continue;
05400
05401     /* Get indices... */
05402     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
05403     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
05404     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
05405
05406     /* Check indices... */
05407     if (ix < 0 || ix >= ctl->prof_nx ||
05408         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
05409         continue;
05410
05411     /* Get total mass in grid cell... */
05412     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
05413 }
05414
05415 /* Extract profiles... */
05416 for (ix = 0; ix < ctl->prof_nx; ix++)
05417     for (iy = 0; iy < ctl->prof_ny; iy++)
05418         if (obscount[ix][iy] >= 1) {
05419
05420             /* Check profile... */
05421             okay = 0;
05422             for (iz = 0; iz < ctl->prof_nz; iz++)
05423                 if (mass[ix][iy][iz] > 0) {
05424                     okay = 1;
05425                     break;
05426                 }
05427             if (!okay)
05428                 continue;
05429
05430             /* Write output... */
05431             fprintf(out, "\n");
05432
05433             /* Loop over altitudes... */
05434             for (iz = 0; iz < ctl->prof_nz; iz++) {
05435
05436                 /* Get pressure and temperature... */
05437                 INTPOL_INIT;
05438                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05439                     lon[ix], lat[iy], &temp, ci, cw, 1);
05440                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
05441                     lon[ix], lat[iy], &h2o, ci, cw, 0);
05442                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
05443                     lon[ix], lat[iy], &o3, ci, cw, 0);
05444
05445                 /* Calculate volume mixing ratio... */
05446                 rho_air = 100. * press[iz] / (RA * temp);
05447                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
05448                     / (rho_air * area[iy] * dz * 1e9);

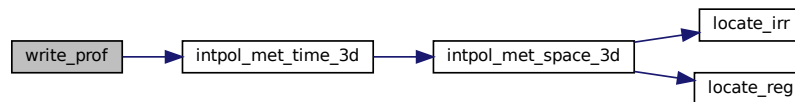
```

```

05449
05450     /* Write output... */
05451     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
05452            t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
05453            obsmean[ix][iy] / obscount[ix][iy], obscount[ix][iy]);
05454     }
05455 }
05456
05457 /* Close files... */
05458 if (t == ctl->t_stop) {
05459     fclose(in);
05460     fclose(out);
05461 }
05462 }

```

Here is the call graph for this function:



5.19.2.51 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 5466 of file libtrac.c.

```

05472     {
05473
05474     static FILE *in, *out;
05475
05476     static char line[LEN];
05477
05478     static double area, dlat, rmax2, t0, t1, rt, rt_old, rz, rlon, rlat, robs;
05479
05480     /* Set timer... */
05481     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
05482
05483     /* Init... */
05484     if (t == ctl->t_start) {
05485
05486         /* Open observation data file... */
05487         LOG(1, "Read sample observation data: %s", ctl->sample_obsfile);
05488         if (!(in = fopen(ctl->sample_obsfile, "r")))
05489             ERRMSG("Cannot open file!");
05490
05491         /* Initialize time for file input... */
05492         rt_old = -1e99;
05493
05494         /* Create new file... */
05495         LOG(1, "Write sample data: %s", filename);
05496         if (!(out = fopen(filename, "w")))
05497             ERRMSG("Cannot create file!");
05498
05499         /* Write header... */
05500         fprintf(out,
05501            "# $1 = time [s]\n"
05502            "# $2 = altitude [km]\n"
05503            "# $3 = longitude [deg]\n"

```



```

05504         "# $4 = latitude [deg]\n"
05505         "# $5 = surface area [km^2]\n"
05506         "# $6 = layer width [km]\n"
05507         "# $7 = number of particles [1]\n"
05508         "# $8 = column density [kg/m^2]\n"
05509         "# $9 = volume mixing ratio [ppv]\n"
05510         "# $10 = observed BT index [K]\n\n");
05511
05512     /* Set latitude range, squared radius, and area... */
05513     dlat = DY2DEG(ctl->sample_dx);
05514     rmax2 = SQR(ctl->sample_dx);
05515     area = M_PI * rmax2;
05516 }
05517
05518 /* Set time interval for output... */
05519 t0 = t - 0.5 * ctl->dt_mod;
05520 t1 = t + 0.5 * ctl->dt_mod;
05521
05522 /* Read observation data... */
05523 while (fgets(line, LEN, in)) {
05524
05525     /* Read data... */
05526     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
05527         5)
05528         continue;
05529
05530     /* Check time... */
05531     if (rt < t0)
05532         continue;
05533     if (rt < rt_old)
05534         ERRMSG("Time must be ascending!");
05535     rt_old = rt;
05536
05537     /* Calculate Cartesian coordinates... */
05538     double x0[3];
05539     geo2cart(0, rln, rln, x0);
05540
05541     /* Set pressure range... */
05542     double rp = P(rz);
05543     double ptop = P(rz + ctl->sample_dz);
05544     double pbot = P(rz - ctl->sample_dz);
05545
05546     /* Init... */
05547     double mass = 0;
05548     int np = 0;
05549
05550     /* Loop over air parcels... */
05551     #pragma omp parallel for default(shared) reduction(+:mass,np)
05552     for (int ip = 0; ip < atm->np; ip++) {
05553
05554         /* Check time... */
05555         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05556             continue;
05557
05558         /* Check latitude... */
05559         if (fabs(rln - atm->lat[ip]) > dlat)
05560             continue;
05561
05562         /* Check horizontal distance... */
05563         double x1[3];
05564         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05565         if (DIST2(x0, x1) > rmax2)
05566             continue;
05567
05568         /* Check pressure... */
05569         if (ctl->sample_dz > 0)
05570             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
05571                 continue;
05572
05573         /* Add mass... */
05574         if (ctl->qnt_m >= 0)
05575             mass += atm->q[ctl->qnt_m][ip];
05576         np++;
05577     }
05578
05579     /* Calculate column density... */
05580     double cd = mass / (1e6 * area);
05581
05582     /* Calculate volume mixing ratio... */
05583     double vmr = 0;
05584     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
05585         if (mass > 0) {
05586
05587             /* Get temperature... */
05588             double temp;
05589             INTPOL_INIT;
05590             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt, rp,

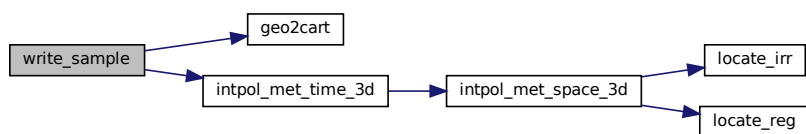
```

```

05591             rlon, rlat, &temp, ci, cw, 1);
05592
05593     /* Calculate density of air... */
05594     double rho_air = 100. * rp / (RA * temp);
05595
05596     /* Calculate volume mixing ratio... */
05597     vmr = MA / ctl->molmass * mass
05598           / (rho_air * 1e6 * area * 1e3 * ctl->sample_dz);
05599 }
05600 } else
05601     vmr = GSL_NAN;
05602
05603 /* Write output... */
05604 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt, rz, rlon, rlat,
05605         area, ctl->sample_dz, np, cd, vmr, robs);
05606
05607 /* Check time... */
05608 if (rt >= tl)
05609     break;
05610 }
05611
05612 /* Close files... */
05613 if (t == ctl->t_stop) {
05614     fclose(in);
05615     fclose(out);
05616 }
05617 }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 5621 of file libtrac.c.

```

05625     {
05626
05627     static FILE *out;
05628
05629     static double rmax2, t0, t1, x0[3], x1[3];
05630
05631     /* Set timer... */
05632     SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
05633
05634     /* Init... */
05635     if (t == ctl->t_start) {
05636
05637         /* Write info... */
05638         LOG(1, "Write station data: %s", filename);
05639
05640         /* Create new file... */
05641         if (!(out = fopen(filename, "w")))
05642             ERRMSG("Cannot create file!");
05643
05644         /* Write header... */
05645         fprintf(out,

```

```

05646         "# $1 = time [s]\n"
05647         "# $2 = altitude [km]\n"
05648         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05649     for (int iq = 0; iq < ctl->nq; iq++)
05650         fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
05651             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05652     fprintf(out, "\n");
05653
05654     /* Set geolocation and search radius... */
05655     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
05656     rmax2 = SQR(ctl->stat_r);
05657 }
05658
05659 /* Set time interval for output... */
05660 t0 = t - 0.5 * ctl->dt_mod;
05661 t1 = t + 0.5 * ctl->dt_mod;
05662
05663 /* Loop over air parcels... */
05664 for (int ip = 0; ip < atm->np; ip++) {
05665
05666     /* Check time... */
05667     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05668         continue;
05669
05670     /* Check time range for station output... */
05671     if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
05672         continue;
05673
05674     /* Check station flag... */
05675     if (ctl->qnt_stat >= 0)
05676         if (atm->q[ctl->qnt_stat][ip])
05677             continue;
05678
05679     /* Get Cartesian coordinates... */
05680     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05681
05682     /* Check horizontal distance... */
05683     if (DIST2(x0, x1) > rmax2)
05684         continue;
05685
05686     /* Set station flag... */
05687     if (ctl->qnt_stat >= 0)
05688         atm->q[ctl->qnt_stat][ip] = 1;
05689
05690     /* Write data... */
05691     fprintf(out, "%.2f %g %g %g",
05692         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
05693     for (int iq = 0; iq < ctl->nq; iq++) {
05694         fprintf(out, " ");
05695         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05696     }
05697     fprintf(out, "\n");
05698 }
05699
05700 /* Close file... */
05701 if (t == ctl->t_stop)
05702     fclose(out);
05703 }

```

Here is the call graph for this function:



5.20 libtrac.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by

```

```

00006 the Free Software Foundation, either version 3 of the License, or
00007 (at your option) any later version.
00008
00009 MPTRAC is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU General Public License for more details.
00013
00014 You should have received a copy of the GNU General Public License
00015 along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*****
00028
00029 void cart2geo(
00030     double *x,
00031     double *z,
00032     double *lon,
00033     double *lat) {
00034
00035     double radius = NORM(x);
00036     *lat = asin(x[2] / radius) * 180. / M_PI;
00037     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00038     *z = radius - RE;
00039 }
00040
00041 /*****
00042
00043 static double clim_hno3_secs[12] = {
00044     1209600.00, 3888000.00, 6393600.00,
00045     9072000.00, 11664000.00, 14342400.00,
00046     16934400.00, 19612800.00, 22291200.00,
00047     24883200.00, 27561600.00, 30153600.00
00048 };
00049
00050 #ifdef _OPENACC
00051 #pragma acc declare copyin(clim_hno3_secs)
00052 #endif
00053
00054 static double clim_hno3_lats[18] = {
00055     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00056     5, 15, 25, 35, 45, 55, 65, 75, 85
00057 };
00058
00059 #ifdef _OPENACC
00060 #pragma acc declare copyin(clim_hno3_lats)
00061 #endif
00062
00063 static double clim_hno3_ps[10] = {
00064     4.64159, 6.81292, 10, 14.678, 21.5443,
00065     31.6228, 46.4159, 68.1292, 100, 146.78
00066 };
00067
00068 #ifdef _OPENACC
00069 #pragma acc declare copyin(clim_hno3_ps)
00070 #endif
00071
00072 static double clim_hno3_var[12][18][10] = {
00073     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00074      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00075      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00076      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00077      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00078      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00079      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00080      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00081      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00082      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00083      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00084      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00085      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00086      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00087      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00088      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00089      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00090      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00091     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00092      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00093      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00094      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00095      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00096      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00097      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},

```

```

00098 {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00099 {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00100 {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00101 {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00102 {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00103 {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00104 {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00105 {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00106 {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00107 {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00108 {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17},
00109 {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00110 {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00111 {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00112 {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00113 {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00114 {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00115 {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00116 {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00117 {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00118 {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00119 {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00120 {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00121 {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00122 {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00123 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00124 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00125 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00126 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42},
00127 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00128 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00129 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00130 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00131 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00132 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00133 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00134 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00135 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00136 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00137 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00138 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00139 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00140 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00141 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00142 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00143 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00144 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62},
00145 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00146 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00147 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00148 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00149 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00150 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00151 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00152 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00153 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00154 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00155 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00156 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00157 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00158 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00159 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00160 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00161 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00162 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00163 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00164 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00165 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00166 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00167 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00168 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00169 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00170 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00171 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00172 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00173 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00174 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00175 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00176 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00177 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00178 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00179 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00180 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},
00181 {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00182 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00183 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00184 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},

```

00185 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00186 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00187 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00188 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00189 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00190 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00191 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00192 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00193 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00194 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00195 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00196 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00197 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00198 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62},
00199 {5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00200 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00201 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00202 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00203 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00204 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00205 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00206 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00207 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00208 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00209 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00210 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00211 {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00212 {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00213 {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00214 {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00215 {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00216 {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55},
00217 {1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00218 {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00219 {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00220 {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00221 {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00222 {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00223 {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00224 {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00225 {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00226 {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00227 {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00228 {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00229 {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00230 {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00231 {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00232 {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00233 {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00234 {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65},
00235 {0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00236 {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00237 {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00238 {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00239 {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00240 {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00241 {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00242 {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00243 {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00244 {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00245 {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00246 {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00247 {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00248 {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00249 {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00250 {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00251 {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00252 {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8},
00253 {0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00254 {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00255 {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00256 {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00257 {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00258 {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00259 {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00260 {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00261 {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00262 {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00263 {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00264 {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00265 {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00266 {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00267 {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00268 {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00269 {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00270 {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05},
00271 {0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},

```

00272     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00273     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00274     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00275     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00276     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00277     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00278     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00279     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00280     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00281     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00282     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00283     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00284     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00285     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00286     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00287     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00288     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00289 };
00290
00291 #ifdef _OPENACC
00292 #pragma acc declare copyin(clim_hno3_var)
00293 #endif
00294
00295 double clim_hno3(
00296     double t,
00297     double lat,
00298     double p) {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Check latitude... */
00312     if (lat < clim_hno3_lats[0])
00313         lat = clim_hno3_lats[0];
00314     else if (lat > clim_hno3_lats[17])
00315         lat = clim_hno3_lats[17];
00316
00317     /* Get indices... */
00318     int isec = locate_irr(clim_hno3_secs, 12, sec);
00319     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00320     int ip = locate_irr(clim_hno3_ps, 10, p);
00321
00322     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00323     double aux00 = LIN(clim_hno3_ps[ip],
00324         clim_hno3_var[isec][ilat][ip],
00325         clim_hno3_ps[ip + 1],
00326         clim_hno3_var[isec][ilat][ip + 1], p);
00327     double aux01 = LIN(clim_hno3_ps[ip],
00328         clim_hno3_var[isec][ilat + 1][ip],
00329         clim_hno3_ps[ip + 1],
00330         clim_hno3_var[isec][ilat + 1][ip + 1], p);
00331     double aux10 = LIN(clim_hno3_ps[ip],
00332         clim_hno3_var[isec + 1][ilat][ip],
00333         clim_hno3_ps[ip + 1],
00334         clim_hno3_var[isec + 1][ilat][ip + 1], p);
00335     double aux11 = LIN(clim_hno3_ps[ip],
00336         clim_hno3_var[isec + 1][ilat + 1][ip],
00337         clim_hno3_ps[ip + 1],
00338         clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00339     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00340         clim_hno3_lats[ilat + 1], aux01, lat);
00341     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00342         clim_hno3_lats[ilat + 1], aux11, lat);
00343     aux00 = LIN(clim_hno3_secs[isec], aux00,
00344         clim_hno3_secs[isec + 1], aux11, sec);
00345
00346     /* Convert from ppb to ppv... */
00347     return GSL_MAX(1e-9 * aux00, 0.0);
00348 }
00349
00350 /*****
00351
00352 static double clim_oh_secs[12] = {
00353     1209600.00, 3888000.00, 6393600.00,
00354     9072000.00, 11664000.00, 14342400.00,
00355     16934400.00, 19612800.00, 22291200.00,
00356     24883200.00, 27561600.00, 30153600.00
00357 };
00358

```

```
00359 #ifdef _OPENACC
00360 #pragma acc declare copyin(clim_oh_secs)
00361 #endif
00362
00363 static double clim_oh_lats[18] = {
00364     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00365     5, 15, 25, 35, 45, 55, 65, 75, 85
00366 };
00367
00368 #ifdef _OPENACC
00369 #pragma acc declare copyin(clim_oh_lats)
00370 #endif
00371
00372 static double clim_oh_ps[34] = {
00373     0.17501, 0.233347, 0.31113, 0.41484, 0.553119, 0.737493, 0.983323,
00374     1.3111, 1.74813, 2.33084, 3.10779, 4.14372, 5.52496, 7.36661, 9.82214,
00375     13.0962, 17.4616, 23.2821, 31.0428, 41.3904, 55.1872, 73.583, 98.1107,
00376     130.814, 174.419, 232.559, 310.078, 413.438, 551.25, 735, 789.809,
00377     848.705, 911.993, 980
00378 };
00379
00380 #ifdef _OPENACC
00381 #pragma acc declare copyin(clim_oh_ps)
00382 #endif
00383
00384 static double clim_oh_var[12][18][34] = {
00385     {{6.422, 6.418, 7.221, 8.409, 9.768, 11.22, 12.65, 13.68, 14.03,
00386        13.06, 11.01, 8.791, 7.096, 6.025, 5.135, 4.057, 2.791, 1.902,
00387        1.318, 0.9553, 0.7083, 0.5542, 0.5145, 0.5485, 0.6292, 0.5982, 1.716,
00388        1.111, 0.9802, 0.6707, 0.5235, 0.4476, 0.3783, 0.3091}},
00389     {6.311, 6.394, 7.2, 8.349, 9.664, 11.02, 12.21, 13.06, 13.28,
00390        12.42, 10.59, 8.552, 6.944, 5.862, 4.948, 3.826, 2.689, 1.873,
00391        1.302, 0.9316, 0.7053, 0.5634, 0.508, 0.5207, 0.6166, 0.6789, 1.682,
00392        1.218, 1.079, 0.7621, 0.6662, 0.5778, 0.4875, 0.3997}},
00393     {5.851, 5.827, 6.393, 7.294, 8.322, 9.415, 10.46, 11.24, 11.59,
00394        11.13, 9.754, 7.97, 6.417, 5.331, 4.468, 3.512, 2.581, 1.855,
00395        1.336, 0.9811, 0.756, 0.6328, 0.6011, 0.6202, 0.7603, 0.8883, 1.303,
00396        1.124, 1.118, 0.9428, 0.8655, 0.8156, 0.7602, 0.6805}},
00397     {5.276, 5.158, 5.66, 6.463, 7.419, 8.488, 9.563, 10.45, 10.94,
00398        10.65, 9.465, 7.762, 6.204, 5.074, 4.209, 3.324, 2.511, 1.865,
00399        1.386, 1.066, 0.8521, 0.723, 0.6997, 0.7492, 0.8705, 0.8088, 1.22,
00400        1.192, 1.298, 1.096, 1.037, 0.9589, 0.8856, 0.7726}},
00401     {5.06, 4.919, 5.379, 6.142, 7.095, 8.156, 9.18, 10.09, 10.62,
00402        10.33, 9.123, 7.479, 5.967, 4.858, 3.987, 3.097, 2.342, 1.743,
00403        1.323, 1.044, 0.8598, 0.7596, 0.7701, 0.7858, 0.8741, 1.256, 1.266,
00404        1.418, 1.594, 1.247, 1.169, 1.111, 1.054, 0.9141}},
00405     {4.921, 4.759, 5.188, 5.936, 6.847, 7.871, 8.903, 9.805, 10.31,
00406        10, 8.818, 7.223, 5.757, 4.66, 3.75, 2.831, 2.1, 1.579,
00407        1.243, 1.017, 0.8801, 0.8193, 0.9409, 1.131, 0.7313, 1.201, 1.383,
00408        1.643, 1.751, 1.494, 1.499, 1.647, 1.934, 2.147}},
00409     {4.665, 4.507, 4.947, 5.652, 6.549, 7.573, 8.609, 9.499, 9.985,
00410        9.664, 8.478, 6.944, 5.519, 4.407, 3.511, 2.595, 1.917, 1.46,
00411        1.172, 1.009, 0.9372, 0.9439, 1.047, 1.219, 0.5712, 1.032, 1.342,
00412        1.716, 1.846, 1.551, 1.55, 1.686, 2.006, 2.235}},
00413     {4.424, 4.288, 4.678, 5.38, 6.271, 7.291, 8.324, 9.231, 9.678,
00414        9.264, 8.037, 6.532, 5.141, 4.037, 3.148, 2.319, 1.715, 1.318,
00415        1.078, 0.9647, 0.9327, 0.9604, 1.023, 0.4157, 0.4762, 1.04, 1.589,
00416        2.093, 1.957, 1.557, 1.52, 1.565, 1.776, 1.904}},
00417     {4.154, 3.996, 4.347, 5.004, 5.854, 6.869, 7.929, 8.837, 9.23,
00418        8.708, 7.447, 6.024, 4.761, 3.742, 2.898, 2.096, 1.55, 1.191,
00419        0.9749, 0.8889, 0.8745, 0.9004, 0.9648, 0.36, 0.4423, 0.973, 1.571,
00420        2.086, 1.971, 1.569, 1.537, 1.567, 1.74, 1.811}},
00421     {3.862, 3.738, 4.093, 4.693, 5.499, 6.481, 7.489, 8.328, 8.637,
00422        8.07, 6.863, 5.56, 4.438, 3.522, 2.736, 1.971, 1.441, 1.098,
00423        0.8945, 0.8155, 0.7965, 0.8013, 0.8582, 1.119, 0.4076, 0.8805, 1.446,
00424        1.977, 1.96, 1.713, 1.793, 2.055, 2.521, 2.776}},
00425     {3.619, 3.567, 3.943, 4.54, 5.295, 6.168, 7.033, 7.691, 7.884,
00426        7.326, 6.207, 5.032, 4.055, 3.263, 2.552, 1.871, 1.365, 1.022,
00427        0.8208, 0.7184, 0.6701, 0.6551, 0.6965, 0.7928, 0.3639, 0.6365, 0.9295,
00428        1.381, 1.847, 1.658, 1.668, 1.87, 2.245, 2.409}},
00429     {3.354, 3.395, 3.811, 4.39, 5.07, 5.809, 6.514, 7, 7.054,
00430        6.472, 5.463, 4.466, 3.649, 2.997, 2.396, 1.785, 1.289, 0.9304,
00431        0.7095, 0.5806, 0.5049, 0.4639, 0.4899, 0.5149, 0.5445, 0.5185, 0.7495,
00432        0.8662, 1.25, 1.372, 1.384, 1.479, 1.76, 1.874}},
00433     {3.008, 3.102, 3.503, 4.049, 4.657, 5.287, 5.845, 6.14, 6.032,
00434        5.401, 4.494, 3.665, 3.043, 2.575, 2.103, 1.545, 1.074, 0.7429,
00435        0.5514, 0.4313, 0.3505, 0.2957, 0.2688, 0.2455, 0.232, 0.3565, 0.4017,
00436        0.5063, 0.6618, 0.7621, 0.7915, 0.8372, 0.923, 0.9218}},
00437     {2.548, 2.725, 3.135, 3.637, 4.165, 4.666, 5.013, 5.056, 4.72,
00438        4.033, 3.255, 2.64, 2.24, 1.942, 1.555, 1.085, 0.7271, 0.502,
00439        0.3748, 0.2897, 0.2303, 0.19, 0.1645, 0.1431, 0.1215, 0.09467, 0.1442,
00440        0.1847, 0.2368, 0.2463, 0.2387, 0.2459, 0.2706, 0.2751}},
00441     {1.946, 2.135, 2.46, 2.831, 3.203, 3.504, 3.584, 3.37, 2.921,
00442        2.357, 1.865, 1.551, 1.392, 1.165, 0.8443, 0.5497, 0.3686, 0.2632,
00443        0.1978, 0.1509, 0.1197, 0.0992, 0.08402, 0.07068, 0.05652, 0.03962,
00444        0.03904,
00445        0.04357, 0.05302, 0.04795, 0.04441, 0.04296, 0.04446, 0.04576},
```



```

00446 {1.157, 1.285, 1.432, 1.546, 1.602, 1.556, 1.378, 1.15, 0.9351,
00447 0.7636, 0.6384, 0.5267, 0.4008, 0.2821, 0.2, 0.1336, 0.09109, 0.06557,
00448 0.05219, 0.04197, 0.03443, 0.03119, 0.02893, 0.02577, 0.02119, 0.01102,
00449 0.008897,
00450 0.00467, 0.004651, 0.004809, 0.004539, 0.004181, 0.003737, 0.002833},
00451 {0.07716, 0.06347, 0.05343, 0.04653, 0.0393, 0.03205, 0.02438, 0.01692,
00452 0.0115,
00453 0.007576, 0.00488, 0.002961, 0.001599, 0.001033, 0.001067, 0.001091,
00454 0.0005156, 0.0003818,
00455 0.0005061, 0.0005322, 0.0008027, 0.0008598, 0.0009114, 0.001112, 0.002042,
00456 0.0002528, 0.0005562,
00457 7.905e-06, 1.461e-07, 1.448e-07, 9.962e-08, 4.304e-08, 9.129e-17,
00458 1.36e-16},
00459 {2.613e-05, 3.434e-05, 3.646e-05, 5.101e-05, 8.027e-05, 0.0001172,
00460 9.886e-05, 1.933e-05, 3.14e-05,
00461 7.708e-05, 0.000136, 0.0001447, 0.0001049, 4.451e-05, 9.37e-06, 2.235e-06,
00462 1.034e-06, 4.87e-06,
00463 1.615e-05, 2.018e-05, 6.578e-05, 0.000178, 0.0002489, 0.0004818, 0.001231,
00464 0.0001402, 0.0004263,
00465 4.581e-07, 1.045e-12, 1.295e-13, 9.008e-14, 8.464e-14, 1.183e-13,
00466 2.471e-13}},
00467 {{5.459, 5.793, 6.743, 7.964, 9.289, 10.5, 11.39, 11.68, 11.14,
00468 9.663, 7.886, 6.505, 5.549, 4.931, 4.174, 3.014, 1.99, 1.339,
00469 0.9012, 0.6096, 0.4231, 0.3152, 0.2701, 0.2561, 0.2696, 0.2523, 0.7171,
00470 0.5333, 0.4876, 0.3218, 0.2536, 0.2178, 0.1861, 0.1546},
00471 {5.229, 5.456, 6.192, 7.112, 8.094, 9.038, 9.776, 10.16, 9.992,
00472 9, 7.493, 6.162, 5.154, 4.461, 3.788, 2.935, 2.058, 1.407,
00473 0.9609, 0.6738, 0.4989, 0.3927, 0.3494, 0.3375, 0.3689, 0.3866, 0.6716,
00474 0.7088, 0.6307, 0.4388, 0.3831, 0.3318, 0.2801, 0.2317},
00475 {4.712, 4.75, 5.283, 6.062, 6.943, 7.874, 8.715, 9.344, 9.516,
00476 8.913, 7.63, 6.223, 5.1, 4.346, 3.709, 2.982, 2.235, 1.605,
00477 1.142, 0.8411, 0.6565, 0.5427, 0.4942, 0.4907, 0.5447, 0.6331, 0.9356,
00478 0.7821, 0.7611, 0.663, 0.628, 0.5915, 0.5763, 0.5451},
00479 {4.621, 4.6, 5.091, 5.827, 6.688, 7.624, 8.529, 9.276, 9.631,
00480 9.219, 7.986, 6.499, 5.264, 4.401, 3.737, 2.996, 2.292, 1.69,
00481 1.237, 0.9325, 0.7325, 0.6093, 0.5742, 0.5871, 0.6446, 0.6139, 0.9845,
00482 0.9741, 1.044, 0.9091, 0.8661, 0.8218, 0.7617, 0.6884},
00483 {4.647, 4.573, 5.038, 5.766, 6.61, 7.534, 8.489, 9.252, 9.6,
00484 9.167, 7.958, 6.512, 5.259, 4.317, 3.547, 2.789, 2.13, 1.601,
00485 1.205, 0.9321, 0.7532, 0.6464, 0.6173, 0.5896, 0.5782, 1.014, 1.096,
00486 1.226, 1.387, 1.111, 1.042, 0.9908, 0.9408, 0.8311},
00487 {4.621, 4.534, 4.984, 5.693, 6.545, 7.49, 8.444, 9.177, 9.531,
00488 9.117, 7.928, 6.533, 5.27, 4.271, 3.431, 2.575, 1.902, 1.42,
00489 1.11, 0.9004, 0.7658, 0.6955, 0.7676, 0.9088, 0.8989, 1.028, 1.221,
00490 1.455, 1.583, 1.375, 1.376, 1.498, 1.744, 1.925},
00491 {4.514, 4.41, 4.837, 5.545, 6.416, 7.38, 8.287, 9.05, 9.416,
00492 9.022, 7.903, 6.496, 5.175, 4.111, 3.232, 2.38, 1.76, 1.335,
00493 1.068, 0.9227, 0.8515, 0.8511, 0.9534, 1.091, 0.4909, 0.9377, 1.241,
00494 1.592, 1.739, 1.478, 1.473, 1.597, 1.893, 2.117},
00495 {4.407, 4.264, 4.61, 5.263, 6.095, 7.046, 8.005, 8.805, 9.201,
00496 8.823, 7.705, 6.299, 4.964, 3.891, 3.046, 2.214, 1.641, 1.261,
00497 1.027, 0.922, 0.8759, 0.8893, 0.9782, 0.3707, 0.4349, 0.976, 1.523,
00498 2.021, 1.906, 1.524, 1.486, 1.53, 1.741, 1.869},
00499 {4.156, 4.007, 4.37, 4.987, 5.777, 6.719, 7.728, 8.578, 8.97,
00500 8.552, 7.409, 6.027, 4.731, 3.684, 2.814, 2.029, 1.501, 1.159,
00501 0.9542, 0.8666, 0.8191, 0.8371, 0.9704, 0.3324, 0.5634, 0.9279, 1.512,
00502 2.042, 1.951, 1.566, 1.535, 1.567, 1.739, 1.822},
00503 {3.98, 3.883, 4.232, 4.841, 5.594, 6.502, 7.497, 8.296, 8.631,
00504 8.161, 7.043, 5.703, 4.51, 3.548, 2.717, 1.951, 1.435, 1.107,
00505 0.9188, 0.8286, 0.772, 0.7564, 0.8263, 0.3026, 0.3854, 0.8743, 1.452,
00506 2.024, 2.012, 1.764, 1.85, 2.125, 2.621, 2.9},
00507 {3.877, 3.811, 4.167, 4.733, 5.462, 6.324, 7.144, 7.862, 8.14,
00508 7.695, 6.613, 5.393, 4.299, 3.42, 2.653, 1.947, 1.44, 1.1,
00509 0.8953, 0.7907, 0.7218, 0.6881, 0.7281, 0.8846, 0.3802, 0.6853, 1.025,
00510 1.593, 2.088, 1.846, 1.866, 2.108, 2.548, 2.737},
00511 {3.636, 3.634, 4.034, 4.637, 5.354, 6.148, 6.899, 7.495, 7.657,
00512 7.138, 6.107, 4.974, 3.983, 3.21, 2.558, 1.932, 1.439, 1.062,
00513 0.8104, 0.6632, 0.5721, 0.517, 0.5182, 0.501, 0.3937, 0.5565, 0.7675,
00514 0.9923, 1.455, 1.554, 1.567, 1.701, 2.077, 2.257},
00515 {3.366, 3.446, 3.885, 4.473, 5.16, 5.9, 6.557, 6.973, 6.973,
00516 6.379, 5.387, 4.385, 3.561, 2.941, 2.428, 1.848, 1.349, 0.9512,
00517 0.6956, 0.5432, 0.4437, 0.3733, 0.3297, 0.2972, 0.2518, 0.4613, 0.5078,
00518 0.6579, 0.8696, 0.9827, 1.028, 1.114, 1.275, 1.315},
00519 {2.975, 3.121, 3.567, 4.146, 4.797, 5.466, 6.003, 6.279, 6.087,
00520 5.372, 4.442, 3.614, 2.986, 2.521, 2.082, 1.532, 1.066, 0.7304,
00521 0.5316, 0.4123, 0.3314, 0.274, 0.2365, 0.2061, 0.1755, 0.1348, 0.2217,
00522 0.2854, 0.387, 0.4315, 0.4301, 0.4585, 0.5195, 0.5349},
00523 {2.434, 2.632, 3.054, 3.577, 4.156, 4.725, 5.1, 5.109, 4.7,
00524 3.958, 3.191, 2.588, 2.182, 1.866, 1.491, 1.034, 0.6936, 0.4774,
00525 0.3551, 0.2749, 0.2203, 0.1829, 0.1544, 0.1325, 0.1103, 0.07716, 0.08626,
00526 0.1037, 0.1455, 0.1418, 0.1351, 0.1339, 0.1407, 0.143},
00527 {1.798, 2.004, 2.333, 2.746, 3.164, 3.49, 3.572, 3.315, 2.833,
00528 2.269, 1.797, 1.485, 1.222, 0.9595, 0.6972, 0.4717, 0.3313, 0.2315,
00529 0.1727, 0.1407, 0.1204, 0.1116, 0.08791, 0.07567, 0.07432, 0.04138,
00530 0.02942,
00531 0.02407, 0.02969, 0.02808, 0.0261, 0.02364, 0.02079, 0.01623},
00532 {1.01, 1.1, 1.194, 1.284, 1.32, 1.278, 1.136, 0.9619, 0.815,

```

00533 0.7226, 0.6136, 0.472, 0.3124, 0.2056, 0.1448, 0.1041, 0.07233, 0.05587,
00534 0.05634, 0.04788, 0.03971, 0.02542, 0.01735, 0.01522, 0.01475, 0.01076,
00535 0.005235,
00536 0.003546, 0.003113, 0.003146, 0.002902, 0.002635, 0.001966, 0.001156},
00537 {0.04181, 0.03949, 0.03577, 0.03191, 0.02839, 0.02391, 0.01796, 0.01251,
00538 0.008422,
00539 0.005359, 0.003257, 0.001769, 0.001092, 0.0008405, 0.001744, 0.0003061,
00540 0.0002956, 0.000334,
00541 0.0004223, 0.0004062, 0.0003886, 0.0003418, 0.0003588, 0.0005709,
00542 0.001368, 0.0001472, 0.0003282,
00543 2.039e-06, 9.891e-13, 1.597e-13, 9.622e-14, 1.167e-13, 1.706e-13,
00544 3.59e-13}},
00545 {{3.361, 3.811, 4.35, 4.813, 5.132, 5.252, 5.019, 4.545, 3.913,
00546 3.25, 2.739, 2.417, 2.071, 1.582, 1.059, 0.6248, 0.3911, 0.2807,
00547 0.2095, 0.1539, 0.118, 0.0953, 0.08084, 0.07456, 0.07567, 0.07108, 0.1196,
00548 0.08475, 0.0839, 0.05724, 0.04673, 0.0412, 0.03626, 0.03134},
00549 {3.193, 3.459, 3.973, 4.578, 5.172, 5.677, 5.945, 5.913, 5.487,
00550 4.686, 3.838, 3.217, 2.773, 2.409, 1.9, 1.33, 0.899, 0.64,
00551 0.4734, 0.355, 0.277, 0.2245, 0.1928, 0.1794, 0.1806, 0.1934, 0.3268,
00552 0.2217, 0.1969, 0.1375, 0.1206, 0.103, 0.08729, 0.07332},
00553 {3.456, 3.617, 4.127, 4.778, 5.448, 6.083, 6.59, 6.858, 6.716,
00554 6.004, 4.975, 4.092, 3.442, 2.986, 2.474, 1.87, 1.328, 0.9231,
00555 0.6642, 0.5044, 0.4019, 0.336, 0.3049, 0.2996, 0.3227, 0.3571, 0.4295,
00556 0.3789, 0.3663, 0.3286, 0.3128, 0.3015, 0.302, 0.2968},
00557 {3.743, 3.817, 4.311, 4.948, 5.666, 6.417, 7.077, 7.528, 7.551,
00558 6.893, 5.766, 4.721, 3.913, 3.326, 2.788, 2.144, 1.555, 1.093,
00559 0.7905, 0.6101, 0.4943, 0.4214, 0.3965, 0.3958, 0.4205, 0.4093, 0.5808,
00560 0.5876, 0.6359, 0.5956, 0.5731, 0.5485, 0.512, 0.4733},
00561 {4.012, 4.012, 4.455, 5.113, 5.866, 6.661, 7.41, 7.98, 8.111,
00562 7.509, 6.364, 5.223, 4.308, 3.602, 2.951, 2.23, 1.625, 1.168,
00563 0.8748, 0.6916, 0.5758, 0.5122, 0.5053, 0.444, 0.4277, 0.7575, 0.7756,
00564 0.8848, 1.034, 0.8886, 0.8352, 0.7965, 0.7562, 0.68},
00565 {4.136, 4.113, 4.549, 5.187, 5.93, 6.754, 7.592, 8.298, 8.546,
00566 8.022, 6.882, 5.668, 4.625, 3.778, 3.016, 2.236, 1.625, 1.193,
00567 0.9253, 0.756, 0.6523, 0.598, 0.6186, 0.6287, 0.4804, 0.8007, 0.9732,
00568 1.175, 1.329, 1.194, 1.193, 1.288, 1.47, 1.603},
00569 {4.185, 4.119, 4.528, 5.186, 5.95, 6.801, 7.678, 8.404, 8.709,
00570 8.272, 7.179, 5.922, 4.767, 3.814, 2.996, 2.187, 1.599, 1.203,
00571 0.9596, 0.8307, 0.7617, 0.7489, 0.822, 0.9684, 0.4374, 0.7664, 1.044,
00572 1.36, 1.525, 1.331, 1.334, 1.447, 1.685, 1.862},
00573 {4.213, 4.111, 4.507, 5.117, 5.882, 6.773, 7.67, 8.464, 8.829,
00574 8.438, 7.378, 6.088, 4.84, 3.802, 2.915, 2.114, 1.561, 1.195,
00575 0.9731, 0.8773, 0.8296, 0.8295, 0.9193, 1.114, 0.3986, 0.8793, 1.394,
00576 1.884, 1.786, 1.437, 1.403, 1.446, 1.64, 1.752},
00577 {4.216, 4.092, 4.458, 5.04, 5.78, 6.67, 7.617, 8.47, 8.883,
00578 8.531, 7.504, 6.163, 4.862, 3.792, 2.891, 2.063, 1.519, 1.174,
00579 0.979, 0.8935, 0.8377, 0.8354, 0.9357, 0.3235, 0.4057, 0.9346, 1.53,
00580 2.078, 1.951, 1.555, 1.525, 1.557, 1.737, 1.825},
00581 {4.168, 4.054, 4.417, 5.011, 5.76, 6.655, 7.601, 8.453, 8.882,
00582 8.539, 7.487, 6.143, 4.853, 3.809, 2.92, 2.088, 1.531, 1.181,
00583 0.9861, 0.8955, 0.8297, 0.8116, 0.8945, 1.137, 0.4003, 0.9318, 1.557,
00584 2.163, 2.095, 1.82, 1.931, 2.246, 2.823, 3.14},
00585 {4.15, 4.078, 4.457, 5.044, 5.791, 6.692, 7.602, 8.343, 8.697,
00586 8.324, 7.282, 5.969, 4.743, 3.764, 2.926, 2.138, 1.572, 1.197,
00587 0.9808, 0.871, 0.7977, 0.764, 0.7663, 0.8665, 0.3969, 0.7472, 1.125,
00588 1.78, 2.277, 1.98, 2.036, 2.365, 2.936, 3.186},
00589 {4.028, 4.006, 4.421, 5.039, 5.793, 6.657, 7.515, 8.216, 8.5,
00590 8.071, 7.01, 5.768, 4.638, 3.752, 3.021, 2.305, 1.714, 1.261,
00591 0.9646, 0.7989, 0.6935, 0.6314, 0.6426, 0.5914, 0.4424, 0.6827, 0.917,
00592 1.27, 1.864, 1.895, 1.922, 2.172, 2.772, 3.141},
00593 {3.825, 3.864, 4.324, 4.97, 5.727, 6.565, 7.363, 7.978, 8.162,
00594 7.628, 6.542, 5.369, 4.372, 3.601, 2.977, 2.292, 1.697, 1.211,
00595 0.8808, 0.6852, 0.562, 0.4717, 0.4094, 0.366, 0.3093, 0.5639, 0.649,
00596 0.8282, 1.144, 1.27, 1.346, 1.522, 1.831, 1.939},
00597 {3.493, 3.611, 4.102, 4.736, 5.491, 6.333, 7.098, 7.615, 7.666,
00598 7.006, 5.88, 4.791, 3.918, 3.283, 2.733, 2.099, 1.522, 1.066,
00599 0.7595, 0.5845, 0.4736, 0.3918, 0.337, 0.2962, 0.2564, 0.2024, 0.3657,
00600 0.4744, 0.6366, 0.76, 0.8059, 0.9015, 1.065, 1.126},
00601 {3.167, 3.353, 3.857, 4.499, 5.225, 6.002, 6.699, 7.106, 6.98,
00602 6.192, 5.094, 4.125, 3.4, 2.87, 2.382, 1.774, 1.246, 0.8549,
00603 0.6089, 0.4666, 0.3762, 0.3082, 0.2643, 0.2283, 0.1942, 0.142, 0.1945,
00604 0.2512, 0.3593, 0.4048, 0.4081, 0.4206, 0.4616, 0.4738},
00605 {2.839, 3.07, 3.565, 4.17, 4.846, 5.529, 6.041, 6.194, 5.833,
00606 5.009, 4.027, 3.239, 2.656, 2.227, 1.794, 1.282, 0.9003, 0.6219,
00607 0.4405, 0.3353, 0.2672, 0.2205, 0.1979, 0.1911, 0.1846, 0.1044, 0.09845,
00608 0.1069, 0.1547, 0.1611, 0.156, 0.1466, 0.1369, 0.1212},
00609 {2.471, 2.75, 3.188, 3.712, 4.27, 4.77, 4.987, 4.802, 4.251,
00610 3.473, 2.731, 2.217, 1.83, 1.498, 1.137, 0.7594, 0.5383, 0.3859,
00611 0.2842, 0.2242, 0.1817, 0.1654, 0.1508, 0.1422, 0.1274, 0.07194, 0.05548,
00612 0.04058, 0.05226, 0.05347, 0.04942, 0.04415, 0.03177, 0.01923},
00613 {2.192, 2.463, 2.735, 2.958, 3.106, 3.134, 2.927, 2.561, 2.149,
00614 1.825, 1.627, 1.429, 1.102, 0.766, 0.4943, 0.3166, 0.2184, 0.1471,
00615 0.1087, 0.08747, 0.07338, 0.06265, 0.05554, 0.05001, 0.04253, 0.03353,
00616 0.01781,
00617 0.01332, 0.01371, 0.01531, 0.01438, 0.01296, 0.00982, 0.006101}},
00618 {{0.2905, 0.282, 0.2639, 0.2366, 0.2078, 0.1879, 0.1644, 0.1407, 0.1163,
00619 0.08898, 0.06233, 0.04005, 0.02438, 0.01319, 0.007679, 0.005681, 0.004482,

```

00620     0.004329,
00621     0.004242, 0.004414, 0.004655, 0.004515, 0.003998, 0.00342, 0.003203,
00622     0.001672, 0.00135,
00623     0.001376, 0.0004656, 0.0003538, 0.000242, 0.0001647, 0.0001078,
00624     5.702e-05},
00625 {1.5, 1.685, 1.901, 2.105, 2.216, 2.203, 2.048, 1.855, 1.609,
00626     1.357, 1.162, 1.012, 0.8042, 0.573, 0.372, 0.223, 0.144, 0.1095,
00627     0.08876, 0.07152, 0.05912, 0.05031, 0.04379, 0.03983, 0.03757, 0.03481,
00628     0.04463,
00629     0.02747, 0.02318, 0.01802, 0.0155, 0.01347, 0.01136, 0.009558},
00630 {2.228, 2.464, 2.889, 3.347, 3.773, 4.098, 4.191, 4.074, 3.7,
00631     3.132, 2.591, 2.217, 1.929, 1.594, 1.177, 0.7799, 0.5288, 0.3786,
00632     0.2805, 0.2144, 0.1695, 0.1403, 0.1236, 0.1157, 0.1134, 0.1148, 0.1127,
00633     0.1059, 0.0982, 0.08934, 0.08645, 0.08666, 0.09037, 0.09339},
00634 {2.761, 2.942, 3.406, 3.981, 4.592, 5.177, 5.591, 5.699, 5.391,
00635     4.682, 3.883, 3.267, 2.804, 2.378, 1.889, 1.347, 0.9255, 0.6382,
00636     0.4661, 0.361, 0.2904, 0.2459, 0.2237, 0.2143, 0.2142, 0.1959, 0.2522,
00637     0.2552, 0.2714, 0.274, 0.2676, 0.2624, 0.2589, 0.2436},
00638 {3.207, 3.317, 3.769, 4.373, 5.05, 5.759, 6.356, 6.677, 6.533,
00639     5.82, 4.861, 4.056, 3.414, 2.865, 2.322, 1.708, 1.204, 0.8386,
00640     0.6195, 0.4861, 0.3997, 0.3459, 0.32, 0.2881, 0.251, 0.4026, 0.4166,
00641     0.4798, 0.5723, 0.5711, 0.5498, 0.5348, 0.5181, 0.4851},
00642 {3.509, 3.549, 3.992, 4.607, 5.301, 6.054, 6.784, 7.302, 7.372,
00643     6.753, 5.706, 4.725, 3.909, 3.223, 2.58, 1.894, 1.356, 0.9716,
00644     0.7397, 0.5944, 0.4998, 0.4401, 0.4193, 0.3649, 0.3204, 0.5317, 0.5992,
00645     0.7541, 0.9257, 0.9193, 0.9167, 0.976, 1.081, 1.167},
00646 {3.748, 3.717, 4.128, 4.736, 5.472, 6.31, 7.086, 7.725, 7.909,
00647     7.387, 6.323, 5.206, 4.217, 3.376, 2.638, 1.912, 1.382, 1.026,
00648     0.8072, 0.6849, 0.6123, 0.5755, 0.6121, 0.6279, 0.3644, 0.5863, 0.823,
00649     1.068, 1.241, 1.134, 1.134, 1.222, 1.397, 1.55},
00650 {3.966, 3.863, 4.239, 4.826, 5.583, 6.465, 7.317, 8.033, 8.34,
00651     7.916, 6.843, 5.63, 4.49, 3.524, 2.698, 1.933, 1.409, 1.074,
00652     0.8747, 0.7756, 0.7211, 0.7103, 0.7513, 0.3147, 0.3627, 0.7582, 1.206,
00653     1.64, 1.639, 1.321, 1.285, 1.334, 1.496, 1.611},
00654 {4.184, 4.066, 4.42, 4.981, 5.712, 6.597, 7.502, 8.305, 8.719,
00655     8.382, 7.314, 6.02, 4.76, 3.702, 2.787, 1.993, 1.461, 1.13,
00656     0.9379, 0.8456, 0.7865, 0.7693, 0.835, 0.3132, 0.4006, 0.9192, 1.486,
00657     1.956, 1.836, 1.485, 1.462, 1.502, 1.65, 1.74},
00658 {4.304, 4.184, 4.566, 5.178, 5.931, 6.823, 7.761, 8.572, 9.013,
00659     8.72, 7.64, 6.283, 4.975, 3.89, 2.959, 2.123, 1.552, 1.194,
00660     0.9949, 0.8938, 0.8268, 0.814, 0.8837, 0.3629, 0.449, 1.027, 1.698,
00661     2.339, 2.232, 1.709, 1.64, 1.636, 1.794, 1.865},
00662 {4.354, 4.279, 4.704, 5.365, 6.153, 7.049, 7.975, 8.793, 9.218,
00663     8.889, 7.805, 6.42, 5.093, 4.013, 3.112, 2.283, 1.671, 1.263,
00664     1.032, 0.9124, 0.8364, 0.8112, 0.8709, 0.9784, 0.5043, 0.9917, 1.587,
00665     2.354, 2.467, 1.85, 1.738, 1.724, 1.873, 1.918},
00666 {4.35, 4.331, 4.785, 5.473, 6.289, 7.199, 8.133, 8.898, 9.288,
00667     8.907, 7.796, 6.411, 5.112, 4.065, 3.212, 2.421, 1.788, 1.331,
00668     1.04, 0.8749, 0.769, 0.7242, 0.7799, 0.8189, 0.5724, 0.86, 1.151,
00669     1.587, 1.946, 1.722, 1.71, 1.845, 2.146, 2.307},
00670 {4.268, 4.264, 4.774, 5.45, 6.298, 7.255, 8.148, 8.907, 9.242,
00671     8.793, 7.612, 6.244, 5.029, 4.124, 3.411, 2.65, 1.98, 1.442,
00672     1.074, 0.8493, 0.7077, 0.6157, 0.5826, 0.5377, 0.4848, 0.7245, 0.8927,
00673     1.153, 1.565, 1.551, 1.638, 1.846, 2.153, 2.304},
00674 {4.119, 4.16, 4.652, 5.372, 6.224, 7.158, 8.06, 8.773, 9.034,
00675     8.507, 7.307, 5.989, 4.866, 4.046, 3.387, 2.637, 1.959, 1.402,
00676     1.01, 0.7722, 0.6266, 0.527, 0.462, 0.4124, 0.3452, 0.6418, 0.572,
00677     0.7352, 1.009, 1.245, 1.364, 1.556, 1.828, 1.914},
00678 {3.932, 4.041, 4.572, 5.285, 6.144, 7.099, 7.985, 8.671, 8.849,
00679     8.203, 6.936, 5.657, 4.621, 3.877, 3.266, 2.526, 1.845, 1.302,
00680     0.9214, 0.6906, 0.5486, 0.4553, 0.4002, 0.3533, 0.3033, 0.24, 0.4062,
00681     0.5084, 0.7133, 0.9299, 0.9714, 1.073, 1.204, 1.211},
00682 {3.882, 4.074, 4.658, 5.414, 6.269, 7.182, 8.05, 8.635, 8.639,
00683     7.828, 6.529, 5.303, 4.339, 3.626, 3.001, 2.262, 1.617, 1.128,
00684     0.7869, 0.574, 0.4488, 0.3713, 0.3216, 0.2794, 0.2403, 0.1962, 0.2638,
00685     0.3018, 0.4483, 0.5618, 0.5717, 0.5812, 0.5993, 0.5956},
00686 {4.195, 4.559, 5.287, 6.13, 7.024, 7.911, 8.623, 8.913, 8.545,
00687     7.44, 6.059, 4.92, 4.063, 3.391, 2.747, 1.953, 1.327, 0.9036,
00688     0.6235, 0.4566, 0.3534, 0.2935, 0.2547, 0.2226, 0.1871, 0.1494, 0.1959,
00689     0.1797, 0.2736, 0.3214, 0.3054, 0.2803, 0.2443, 0.1504},
00690 {4.644, 5.211, 6.192, 7.373, 8.534, 9.582, 10.35, 10.37, 9.49,
00691     7.907, 6.299, 5.088, 4.224, 3.492, 2.701, 1.795, 1.184, 0.8001,
00692     0.5348, 0.3789, 0.2861, 0.2435, 0.2055, 0.1754, 0.1605, 0.1076, 0.1644,
00693     0.09069, 0.1624, 0.2356, 0.223, 0.2001, 0.164, 0.09807}},
00694 {{0.0001362, 0.0001753, 0.0001792, 0.0001881, 0.0002233, 0.0002056,
00695     0.0001083, 1.505e-05, 1.073e-05,
00696     1.22e-05, 7.475e-06, 3.518e-06, 1.292e-06, 3.812e-07, 8.611e-08,
00697     3.148e-09, 1.29e-09, 2.111e-08,
00698     7.591e-07, 1.125e-06, 4.905e-06, 1.473e-05, 3.597e-05, 9.35e-05,
00699     0.0002623, 0.0001066, 0.0002064,
00700     0.0001924, 6.532e-10, 3.205e-10, 3.838e-10, 3.541e-10, 3.331e-10,
00701     2.774e-10},
00702 {0.1865, 0.1757, 0.1596, 0.1454, 0.1343, 0.125, 0.1033, 0.08324, 0.06202,
00703     0.04341, 0.0295, 0.0187, 0.01129, 0.007115, 0.004932, 0.003824, 0.002733,
00704     0.003019,
00705     0.004022, 0.004428, 0.004414, 0.003826, 0.003009, 0.002117, 0.001375,
00706     0.0009066, 0.001031,

```

```
00707 0.0004673, 0.0002194, 0.0001683, 0.0001154, 6.607e-05, 3.128e-05,
00708 1.601e-05},
00709 {1.238, 1.384, 1.576, 1.788, 1.95, 1.993, 1.845, 1.593, 1.329,
00710 1.098, 0.9387, 0.8174, 0.6531, 0.4579, 0.2899, 0.1818, 0.1253, 0.09461,
00711 0.0741, 0.05946, 0.04989, 0.04353, 0.03862, 0.03469, 0.03181, 0.02862,
00712 0.02499,
00713 0.02062, 0.0178, 0.01681, 0.0162, 0.01679, 0.01921, 0.02192},
00714 {1.92, 2.106, 2.478, 2.931, 3.421, 3.86, 4.043, 3.866, 3.41,
00715 2.84, 2.372, 2.077, 1.825, 1.472, 1.07, 0.7023, 0.4688, 0.3296,
00716 0.2453, 0.1889, 0.15, 0.1246, 0.1094, 0.09788, 0.09079, 0.08267, 0.08977,
00717 0.09461, 0.09867, 0.09977, 0.09675, 0.09757, 0.1011, 0.1004},
00718 {2.505, 2.661, 3.095, 3.659, 4.295, 4.928, 5.368, 5.413, 5.018,
00719 4.291, 3.582, 3.052, 2.636, 2.193, 1.714, 1.199, 0.8121, 0.5616,
00720 0.4193, 0.3284, 0.2656, 0.2245, 0.2009, 0.1819, 0.1615, 0.2142, 0.2142,
00721 0.2523, 0.3029, 0.3253, 0.3248, 0.323, 0.3186, 0.3005},
00722 {3.053, 3.153, 3.582, 4.16, 4.823, 5.525, 6.111, 6.38, 6.176,
00723 5.453, 4.575, 3.84, 3.219, 2.652, 2.09, 1.509, 1.055, 0.7478,
00724 0.5678, 0.4529, 0.3746, 0.322, 0.2979, 0.2602, 0.284, 0.3515, 0.39,
00725 0.4915, 0.6421, 0.6745, 0.684, 0.7208, 0.7815, 0.8179},
00726 {3.422, 3.44, 3.875, 4.477, 5.191, 5.965, 6.661, 7.123, 7.132,
00727 6.48, 5.463, 4.526, 3.718, 3.005, 2.344, 1.688, 1.206, 0.8837,
00728 0.6856, 0.5649, 0.4863, 0.445, 0.4856, 0.5006, 0.3136, 0.4515, 0.6109,
00729 0.7863, 0.9684, 0.937, 0.937, 0.9949, 1.106, 1.204},
00730 {3.782, 3.708, 4.085, 4.684, 5.42, 6.264, 7.105, 7.757, 7.965,
00731 7.421, 6.309, 5.169, 4.161, 3.312, 2.538, 1.814, 1.314, 0.9921,
00732 0.7986, 0.6906, 0.6241, 0.6037, 0.6883, 0.3135, 0.3539, 0.6781, 1.044,
00733 1.41, 1.465, 1.198, 1.162, 1.195, 1.315, 1.399},
00734 {4.093, 3.956, 4.303, 4.881, 5.64, 6.546, 7.473, 8.261, 8.617,
00735 8.211, 7.109, 5.806, 4.611, 3.616, 2.718, 1.923, 1.396, 1.077,
00736 0.8999, 0.7993, 0.7318, 0.69, 0.7736, 0.3307, 0.5444, 0.8973, 1.424,
00737 1.867, 1.76, 1.423, 1.397, 1.429, 1.558, 1.631},
00738 {4.316, 4.179, 4.572, 5.214, 6.033, 6.982, 7.93, 8.754, 9.154,
00739 8.809, 7.719, 6.283, 4.962, 3.898, 2.961, 2.106, 1.526, 1.178,
00740 0.9911, 0.8826, 0.8091, 0.792, 0.8984, 0.4081, 0.6591, 1.079, 1.762,
00741 2.394, 2.262, 1.722, 1.649, 1.64, 1.793, 1.856},
00742 {4.422, 4.322, 4.787, 5.478, 6.345, 7.323, 8.254, 9.118, 9.564,
00743 9.218, 8.075, 6.604, 5.203, 4.058, 3.117, 2.274, 1.677, 1.284,
00744 1.048, 0.9151, 0.8297, 0.7966, 0.8935, 0.5312, 0.7531, 1.072, 1.715,
00745 2.495, 2.546, 1.897, 1.782, 1.77, 1.93, 1.983},
00746 {4.547, 4.469, 4.942, 5.688, 6.549, 7.488, 8.486, 9.332, 9.775,
00747 9.417, 8.264, 6.785, 5.379, 4.249, 3.33, 2.505, 1.852, 1.39,
00748 1.098, 0.9308, 0.8263, 0.7895, 0.8576, 1.037, 0.635, 0.9803, 1.371,
00749 1.867, 2.133, 1.823, 1.82, 1.997, 2.372, 2.584},
00750 {4.687, 4.634, 5.104, 5.806, 6.687, 7.693, 8.664, 9.479, 9.89,
00751 9.543, 8.412, 6.918, 5.518, 4.426, 3.541, 2.711, 2.014, 1.493,
00752 1.144, 0.9317, 0.797, 0.7099, 0.6575, 0.6067, 0.526, 0.9154, 1.118,
00753 1.507, 1.832, 1.788, 1.887, 2.179, 2.647, 2.868},
00754 {4.695, 4.698, 5.185, 5.955, 6.871, 7.893, 8.877, 9.693, 10.1,
00755 9.723, 8.528, 6.981, 5.591, 4.567, 3.788, 2.967, 2.242, 1.655,
00756 1.222, 0.9406, 0.7599, 0.6428, 0.57, 0.513, 0.4496, 0.9249, 0.8597,
00757 1.064, 1.428, 1.661, 1.831, 2.151, 2.551, 2.772},
00758 {4.817, 4.823, 5.364, 6.154, 7.109, 8.194, 9.225, 10.09, 10.53,
00759 10.13, 8.82, 7.204, 5.79, 4.774, 3.99, 3.147, 2.374, 1.734,
00760 1.249, 0.9272, 0.7213, 0.5919, 0.522, 0.4691, 0.4205, 0.3435, 0.704,
00761 0.795, 1.078, 1.369, 1.506, 1.697, 1.921, 2.057},
00762 {5.178, 5.275, 5.904, 6.794, 7.806, 8.901, 9.941, 10.76, 11.06,
00763 10.41, 8.899, 7.194, 5.78, 4.767, 3.992, 3.132, 2.304, 1.622,
00764 1.125, 0.8041, 0.6135, 0.4964, 0.4334, 0.388, 0.3494, 0.3124, 0.5608,
00765 0.6202, 0.8549, 1.142, 1.185, 1.327, 1.469, 1.521},
00766 {5.765, 5.982, 6.854, 7.999, 9.258, 10.55, 11.75, 12.58, 12.65,
00767 11.51, 9.567, 7.638, 6.122, 5.016, 4.17, 3.208, 2.283, 1.528,
00768 0.9814, 0.6677, 0.4925, 0.3884, 0.3284, 0.2876, 0.2571, 0.2207, 0.5107,
00769 0.5167, 0.7005, 0.8149, 0.7878, 0.7521, 0.6899, 0.4697},
00770 {5.862, 6.014, 6.876, 8.048, 9.368, 10.8, 12.34, 13.65, 13.83,
00771 12.41, 10.19, 8.106, 6.468, 5.285, 4.422, 3.392, 2.324, 1.509,
00772 0.9796, 0.6834, 0.5019, 0.387, 0.3295, 0.2804, 0.2425, 0.2084, 0.4694,
00773 0.4497, 0.5949, 0.6504, 0.6155, 0.5696, 0.5077, 0.2744}},
00774 {{6.617e-05, 8.467e-05, 8.509e-05, 9.824e-05, 0.0001317, 0.0001499,
00775 0.0001104, 1.226e-05, 1.003e-05,
00776 1.345e-05, 8.036e-06, 2.28e-06, 3.166e-07, 1.803e-08, 6.079e-10,
00777 3.031e-10, 1.336e-09, 1.748e-08,
00778 3.057e-07, 8.184e-07, 1.95e-06, 2.238e-06, 4.658e-06, 1.393e-05,
00779 4.326e-05, 3.288e-05, 0.0001662,
00780 8.012e-05, 6.48e-10, 3.371e-10, 4.509e-10, 4.052e-10, 3.677e-10,
00781 2.996e-10},
00782 {0.003936, 0.002158, 0.001688, 0.001555, 0.001396, 0.0008637, 0.0003091,
00783 4.908e-05, 1.173e-05,
00784 1.326e-05, 9.097e-06, 3.324e-06, 6.336e-07, 8.553e-08, 4.851e-09,
00785 3.058e-09, 9.169e-09, 2.451e-07,
00786 8.77e-07, 2.973e-07, 2.556e-07, 2.79e-07, 8.674e-07, 4.17e-06, 2.373e-05,
00787 3.876e-05, 6.493e-05,
00788 2.13e-05, 2.574e-09, 2.676e-10, 2.356e-10, 1.947e-10, 2.376e-10,
00789 1.955e-10},
00790 {0.8642, 0.927, 0.9782, 1.001, 0.9705, 0.885, 0.7521, 0.6223, 0.5357,
00791 0.4614, 0.373, 0.2955, 0.2138, 0.1342, 0.07775, 0.04773, 0.0348, 0.02871,
00792 0.02503, 0.02329, 0.02308, 0.02227, 0.01967, 0.01687, 0.01381, 0.008063,
00793 0.006489,
```

```

00794    0.005433, 0.004804, 0.00487, 0.004735, 0.004696, 0.004587, 0.003634},
00795 {1.584, 1.731, 2.006, 2.368, 2.742, 3.047, 3.124, 2.894, 2.517,
00796 2.087, 1.761, 1.588, 1.399, 1.091, 0.7549, 0.4844, 0.3282, 0.2357,
00797 0.1737, 0.1299, 0.1001, 0.08162, 0.0699, 0.06056, 0.05333, 0.07014,
00798 0.04552,
00799 0.04907, 0.04781, 0.04938, 0.04864, 0.04971, 0.05141, 0.05078},
00800 {2.256, 2.4, 2.786, 3.287, 3.854, 4.41, 4.773, 4.776, 4.396,
00801 3.755, 3.16, 2.747, 2.381, 1.94, 1.458, 0.9865, 0.6576, 0.4582,
00802 0.3447, 0.2672, 0.2121, 0.1768, 0.1538, 0.1358, 0.1219, 0.1466, 0.1371,
00803 0.161, 0.1896, 0.2116, 0.2133, 0.2153, 0.2145, 0.2042},
00804 {2.844, 2.913, 3.296, 3.841, 4.473, 5.127, 5.672, 5.933, 5.724,
00805 5.005, 4.187, 3.549, 3.006, 2.486, 1.928, 1.368, 0.9363, 0.6524,
00806 0.4946, 0.3921, 0.3189, 0.2696, 0.2503, 0.2181, 0.2227, 0.2946, 0.297,
00807 0.3704, 0.4834, 0.5473, 0.5565, 0.581, 0.6189, 0.6392},
00808 {3.309, 3.337, 3.765, 4.35, 5.025, 5.756, 6.438, 6.908, 6.947,
00809 6.3, 5.252, 4.33, 3.576, 2.912, 2.258, 1.618, 1.15, 0.8271,
00810 0.6351, 0.5129, 0.4287, 0.3846, 0.4351, 0.4889, 0.3229, 0.4124, 0.4839,
00811 0.6285, 0.8468, 0.9091, 0.9059, 0.9413, 1.007, 1.074},
00812 {3.725, 3.662, 4.042, 4.647, 5.381, 6.204, 7.019, 7.636, 7.841,
00813 7.318, 6.177, 5.04, 4.09, 3.299, 2.545, 1.822, 1.303, 0.967,
00814 0.7683, 0.6506, 0.5816, 0.5712, 0.667, 0.3774, 0.3833, 0.5497, 0.7343,
00815 0.9899, 1.25, 1.213, 1.147, 1.144, 1.27, 1.389},
00816 {4.074, 3.916, 4.273, 4.851, 5.624, 6.564, 7.525, 8.296, 8.632,
00817 8.184, 7.013, 5.711, 4.593, 3.691, 2.813, 2.003, 1.445, 1.097,
00818 0.8982, 0.7815, 0.7103, 0.6929, 0.7841, 0.3697, 0.42, 0.8027, 1.255,
00819 1.716, 1.727, 1.411, 1.379, 1.416, 1.548, 1.618},
00820 {4.378, 4.233, 4.657, 5.297, 6.121, 7.098, 8.11, 8.937, 9.31,
00821 8.877, 7.696, 6.289, 5.029, 3.997, 3.066, 2.189, 1.589, 1.209,
00822 0.9953, 0.8642, 0.7883, 0.7739, 0.9011, 0.4115, 0.6312, 0.9904, 1.555,
00823 2.033, 1.915, 1.547, 1.523, 1.55, 1.675, 1.75},
00824 {4.574, 4.464, 4.871, 5.569, 6.471, 7.487, 8.486, 9.356, 9.788,
00825 9.387, 8.196, 6.717, 5.357, 4.245, 3.29, 2.387, 1.738, 1.316,
00826 1.064, 0.9101, 0.8224, 0.8068, 0.9022, 0.478, 0.7124, 1.093, 1.75,
00827 2.355, 2.233, 1.721, 1.663, 1.652, 1.836, 1.906},
00828 {4.771, 4.625, 5.041, 5.796, 6.698, 7.684, 8.72, 9.611, 10.08,
00829 9.736, 8.573, 7.013, 5.571, 4.447, 3.501, 2.606, 1.915, 1.439,
00830 1.139, 0.956, 0.8517, 0.8306, 0.9424, 1.177, 0.5869, 1.036, 1.44,
00831 1.878, 2.039, 1.705, 1.737, 1.923, 2.385, 2.641},
00832 {5.006, 4.866, 5.324, 6.09, 7.006, 8.036, 9.084, 9.893, 10.33,
00833 10.02, 8.839, 7.228, 5.736, 4.611, 3.71, 2.8, 2.063, 1.532,
00834 1.183, 0.9673, 0.8363, 0.7853, 0.8359, 0.8923, 0.569, 1.039, 1.365,
00835 1.742, 1.904, 1.758, 1.875, 2.251, 2.843, 3.224},
00836 {5.099, 4.961, 5.447, 6.211, 7.152, 8.22, 9.322, 10.2, 10.72,
00837 10.48, 9.248, 7.559, 5.993, 4.842, 3.975, 3.116, 2.35, 1.736,
00838 1.297, 1.003, 0.8134, 0.702, 0.6582, 0.6213, 0.5842, 0.5417, 1.158,
00839 1.4, 1.637, 1.863, 2.063, 2.497, 3.122, 3.523},
00840 {5.329, 5.217, 5.717, 6.57, 7.585, 8.724, 9.852, 10.78, 11.29,
00841 10.97, 9.671, 7.914, 6.3, 5.116, 4.263, 3.389, 2.58, 1.892,
00842 1.383, 1.036, 0.8131, 0.6799, 0.6149, 0.5698, 0.5283, 0.4671, 1.093,
00843 1.188, 1.482, 1.733, 1.908, 2.209, 2.612, 2.814},
00844 {6.071, 6.113, 6.75, 7.703, 8.826, 10.05, 11.16, 11.96, 12.25,
00845 11.61, 10.02, 8.102, 6.438, 5.24, 4.366, 3.448, 2.553, 1.795,
00846 1.245, 0.8829, 0.6737, 0.5561, 0.4996, 0.4579, 0.4257, 0.4016, 0.968,
00847 1.042, 1.284, 1.504, 1.639, 1.849, 2.098, 2.255},
00848 {6.417, 6.432, 7.23, 8.422, 9.846, 11.4, 12.82, 13.78, 14,
00849 12.98, 10.94, 8.756, 6.951, 5.69, 4.77, 3.716, 2.672, 1.818,
00850 1.224, 0.8517, 0.6496, 0.5355, 0.4766, 0.4254, 0.389, 0.351, 0.9729,
00851 0.9755, 1.108, 1.026, 0.9967, 0.9687, 0.9329, 0.8183},
00852 {6.462, 6.445, 7.221, 8.396, 9.796, 11.38, 12.96, 14.19, 14.59,
00853 13.52, 11.33, 9.046, 7.189, 5.92, 5.015, 3.949, 2.77, 1.841,
00854 1.239, 0.86, 0.6441, 0.5179, 0.446, 0.42, 0.3818, 0.3068, 0.9267,
00855 0.8849, 0.9828, 0.8458, 0.7544, 0.6963, 0.6233, 0.5358},
00856 {{3.24e-05, 4.086e-05, 4.221e-05, 5.381e-05, 8.808e-05, 0.0001261,
00857 0.0001263, 2.932e-05, 2.057e-05,
00858 4.321e-05, 3.43e-05, 1.213e-05, 1.862e-06, 1.067e-07, 2.983e-09,
00859 2.578e-08, 2.254e-09, 1.504e-08,
00860 1.599e-07, 1.523e-07, 2.033e-07, 5.271e-07, 1.417e-06, 4.518e-06,
00861 1.358e-05, 9.266e-06, 0.0001569,
00862 3.355e-05, 7.653e-10, 3.889e-10, 5.082e-10, 4.498e-10, 4.072e-10,
00863 3.302e-10},
00864 {0.0688, 0.05469, 0.04502, 0.03604, 0.02861, 0.02128, 0.01453, 0.00924,
00865 0.00574,
00866 0.003447, 0.001888, 0.001147, 0.001154, 0.001548, 0.001193, 6.955e-05,
00867 0.0003258, 1.673e-05,
00868 1.393e-05, 1.849e-05, 2.841e-05, 4.045e-05, 5.864e-05, 4.426e-05,
00869 1.722e-05, 3.186e-05, 5.582e-05,
00870 6.823e-06, 3.559e-09, 3.282e-10, 2.847e-10, 2.3e-10, 2.728e-10,
00871 2.237e-10},
00872 {1.056, 1.158, 1.28, 1.377, 1.429, 1.395, 1.242, 1.043, 0.8709,
00873 0.7435, 0.6183, 0.4824, 0.3665, 0.2668, 0.1749, 0.098, 0.09722, 0.08524,
00874 0.05545, 0.04501, 0.04476, 0.04491, 0.04231, 0.03868, 0.03336, 0.01108,
00875 0.01596,
00876 0.009172, 0.008499, 0.008763, 0.008672, 0.008872, 0.008926, 0.007552},
00877 {1.798, 1.952, 2.239, 2.589, 2.958, 3.261, 3.328, 3.12, 2.741,
00878 2.293, 1.917, 1.686, 1.53, 1.28, 0.9362, 0.6052, 0.4093, 0.294,
00879 0.2166, 0.1609, 0.1206, 0.09532, 0.08094, 0.07007, 0.06252, 0.1111,
00880 0.04996,

```

00881 0.0572, 0.0583, 0.06137, 0.06077, 0.0616, 0.0644, 0.06454},
00882 {2.42, 2.566, 2.952, 3.45, 3.987, 4.499, 4.853, 4.878, 4.55,
00883 3.918, 3.264, 2.802, 2.467, 2.094, 1.632, 1.127, 0.7523, 0.5168,
00884 0.386, 0.2982, 0.2346, 0.1928, 0.1694, 0.1534, 0.1384, 0.1672, 0.1434,
00885 0.1726, 0.2115, 0.2414, 0.2445, 0.2487, 0.2496, 0.2381},
00886 {3.026, 3.096, 3.501, 4.016, 4.608, 5.244, 5.786, 6.088, 5.975,
00887 5.326, 4.429, 3.701, 3.142, 2.636, 2.071, 1.477, 1.015, 0.7056,
00888 0.5315, 0.4215, 0.3413, 0.2869, 0.2681, 0.2507, 0.2616, 0.3004, 0.3042,
00889 0.3879, 0.5203, 0.6045, 0.6196, 0.651, 0.6986, 0.7171},
00890 {3.463, 3.475, 3.878, 4.441, 5.112, 5.838, 6.522, 7.015, 7.096,
00891 6.52, 5.487, 4.496, 3.734, 3.105, 2.456, 1.776, 1.26, 0.9016,
00892 0.6841, 0.5485, 0.454, 0.404, 0.4609, 0.5432, 0.376, 0.4702, 0.5045,
00893 0.6555, 0.9157, 1.076, 1.113, 1.216, 1.372, 1.497},
00894 {3.745, 3.679, 4.071, 4.666, 5.394, 6.22, 7.064, 7.725, 7.961,
00895 7.448, 6.321, 5.161, 4.233, 3.472, 2.73, 1.96, 1.409, 1.04,
00896 0.8213, 0.6863, 0.6068, 0.5872, 0.7122, 0.9507, 0.4302, 0.5845, 0.768,
00897 1.036, 1.308, 1.372, 1.393, 1.57, 2.023, 2.387},
00898 {4.046, 3.927, 4.286, 4.869, 5.645, 6.599, 7.567, 8.4, 8.74,
00899 8.247, 7.036, 5.735, 4.638, 3.741, 2.905, 2.066, 1.493, 1.132,
00900 0.9307, 0.7993, 0.7306, 0.7334, 0.8811, 1.197, 0.4369, 0.7932, 1.239,
00901 1.716, 1.764, 1.639, 1.738, 2.042, 2.543, 2.814},
00902 {4.303, 4.169, 4.566, 5.224, 6.055, 7.053, 8.124, 9.037, 9.471,
00903 8.998, 7.714, 6.277, 4.981, 3.935, 3.018, 2.182, 1.599, 1.224,
00904 1.01, 0.8634, 0.7916, 0.7955, 0.9234, 1.237, 0.6362, 0.9588, 1.513,
00905 1.996, 1.902, 1.545, 1.53, 1.572, 1.712, 1.768},
00906 {4.506, 4.381, 4.819, 5.529, 6.428, 7.462, 8.518, 9.453, 9.907,
00907 9.467, 8.237, 6.703, 5.29, 4.149, 3.213, 2.346, 1.732, 1.324,
00908 1.075, 0.9181, 0.8397, 0.8375, 0.9434, 0.5018, 0.5414, 1.083, 1.74,
00909 2.347, 2.227, 1.721, 1.661, 1.648, 1.831, 1.912},
00910 {4.687, 4.558, 4.978, 5.725, 6.658, 7.731, 8.789, 9.675, 10.13,
00911 9.748, 8.526, 6.958, 5.498, 4.351, 3.431, 2.537, 1.87, 1.415,
00912 1.127, 0.9521, 0.8587, 0.8423, 0.8885, 1.184, 0.5938, 1.064, 1.469,
00913 1.877, 2.006, 1.669, 1.7, 1.879, 2.347, 2.626},
00914 {4.936, 4.809, 5.237, 6.008, 6.937, 7.97, 9.016, 9.924, 10.41,
00915 10.04, 8.758, 7.153, 5.667, 4.529, 3.633, 2.737, 2.011, 1.493,
00916 1.161, 0.9601, 0.8439, 0.8078, 0.8976, 1.023, 0.6551, 1.239, 1.595,
00917 1.9, 1.923, 1.721, 1.833, 2.227, 2.857, 3.283},
00918 {5.075, 4.962, 5.428, 6.241, 7.2, 8.253, 9.328, 10.2, 10.65,
00919 10.28, 8.983, 7.308, 5.772, 4.63, 3.752, 2.906, 2.151, 1.57,
00920 1.188, 0.9525, 0.8028, 0.7168, 0.7021, 0.7201, 0.7339, 0.6036, 1.423,
00921 1.574, 1.715, 1.841, 2.05, 2.525, 3.214, 3.663},
00922 {5.33, 5.234, 5.731, 6.529, 7.518, 8.638, 9.699, 10.59, 11.06,
00923 10.69, 9.363, 7.602, 6.028, 4.875, 4.029, 3.2, 2.426, 1.77,
00924 1.305, 0.9937, 0.7977, 0.6831, 0.6247, 0.6058, 0.5859, 0.5171, 1.306,
00925 1.309, 1.552, 1.652, 1.83, 2.179, 2.619, 2.844},
00926 {5.866, 5.848, 6.455, 7.346, 8.396, 9.533, 10.53, 11.34, 11.68,
00927 11.13, 9.619, 7.806, 6.227, 5.073, 4.235, 3.4, 2.554, 1.805,
00928 1.264, 0.9221, 0.7291, 0.623, 0.5765, 0.5499, 0.5313, 0.5058, 1.13,
00929 1.193, 1.384, 1.521, 1.658, 1.903, 2.204, 2.394},
00930 {6.303, 6.398, 7.222, 8.351, 9.705, 11.15, 12.37, 13.21, 13.37,
00931 12.4, 10.47, 8.393, 6.679, 5.472, 4.597, 3.653, 2.67, 1.827,
00932 1.234, 0.8789, 0.697, 0.5977, 0.5427, 0.5065, 0.4765, 0.4416, 1.108,
00933 1.122, 1.2, 1.026, 0.9995, 0.9762, 0.9406, 0.8155},
00934 {6.374, 6.361, 7.22, 8.387, 9.771, 11.27, 12.69, 13.72, 14.07,
00935 13.07, 10.88, 8.681, 6.947, 5.79, 4.913, 3.885, 2.773, 1.855,
00936 1.244, 0.9142, 0.7256, 0.598, 0.5151, 0.5035, 0.4732, 0.3585, 1.059,
00937 1.045, 1.089, 0.8483, 0.7697, 0.7077, 0.6288, 0.5436},
00938 {{0.03789, 0.03408, 0.03134, 0.02846, 0.02395, 0.01876, 0.01296, 0.008683,
00939 0.005595,
00940 0.003327, 0.001899, 0.001016, 0.0006645, 0.001147, 0.000686, 0.0006987,
00941 0.0006744, 0.0003903,
00942 1.823e-05, 1.418e-05, 1.097e-05, 9.197e-06, 8.385e-06, 5.361e-06,
00943 1.047e-05, 8.724e-06, 0.0001224,
00944 1.608e-05, 7.761e-10, 4.005e-10, 5.216e-10, 4.653e-10, 4.282e-10,
00945 3.472e-10},
00946 {0.9685, 1.035, 1.103, 1.161, 1.165, 1.106, 0.9814, 0.8435, 0.7222,
00947 0.6291, 0.5477, 0.4518, 0.3058, 0.2105, 0.1492, 0.08369, 0.1841, 0.1379,
00948 0.1032, 0.06085, 0.03194, 0.02288, 0.0206, 0.02385, 0.0298, 0.0102,
00949 0.002722,
00950 0.00743, 0.007294, 0.006871, 0.006275, 0.005435, 0.004153, 0.002587},
00951 {1.842, 2.002, 2.295, 2.618, 2.95, 3.205, 3.226, 2.966, 2.55,
00952 2.078, 1.668, 1.355, 1.079, 0.8142, 0.7131, 0.5158, 0.3251, 0.2412,
00953 0.1924, 0.1651, 0.1599, 0.164, 0.1325, 0.1215, 0.09591, 0.1117, 0.03993,
00954 0.04814, 0.04778, 0.04729, 0.04627, 0.0467, 0.04597, 0.04172},
00955 {2.489, 2.643, 3.032, 3.507, 4.035, 4.533, 4.829, 4.772, 4.375,
00956 3.714, 3.06, 2.551, 2.207, 1.938, 1.628, 1.185, 0.8109, 0.5625,
00957 0.4101, 0.3078, 0.2325, 0.1813, 0.1507, 0.1311, 0.1178, 0.1964, 0.1063,
00958 0.1261, 0.1425, 0.1561, 0.1546, 0.1532, 0.1543, 0.1498},
00959 {3.03, 3.156, 3.579, 4.144, 4.775, 5.42, 5.935, 6.129, 5.878,
00960 5.138, 4.235, 3.481, 2.956, 2.542, 2.086, 1.502, 1.028, 0.7057,
00961 0.5223, 0.4097, 0.3291, 0.2729, 0.2417, 0.2235, 0.2035, 0.2565, 0.2228,
00962 0.274, 0.339, 0.387, 0.3918, 0.396, 0.3931, 0.3692},
00963 {3.447, 3.508, 3.934, 4.533, 5.226, 5.96, 6.611, 7.003, 6.968,
00964 6.331, 5.301, 4.324, 3.576, 3.009, 2.45, 1.808, 1.285, 0.894,
00965 0.6612, 0.5244, 0.4273, 0.3588, 0.3341, 0.3096, 0.296, 0.4036, 0.403,
00966 0.5114, 0.6763, 0.7735, 0.7969, 0.8461, 0.9229, 0.9621},
00967 {3.722, 3.706, 4.103, 4.704, 5.429, 6.225, 6.997, 7.531, 7.688,

00968 7.197, 6.151, 5.021, 4.119, 3.436, 2.786, 2.084, 1.513, 1.082,
00969 0.808, 0.6428, 0.5308, 0.473, 0.5475, 0.6436, 0.4784, 0.5735, 0.5953,
00970 0.7959, 1.106, 1.287, 1.339, 1.478, 1.689, 1.86},
00971 {3.921, 3.841, 4.189, 4.783, 5.529, 6.4, 7.263, 7.978, 8.293,
00972 7.875, 6.788, 5.549, 4.498, 3.668, 2.902, 2.117, 1.551, 1.158,
00973 0.9124, 0.7546, 0.6628, 0.6526, 0.7867, 0.9666, 0.4912, 0.664, 0.8688,
00974 1.172, 1.475, 1.537, 1.57, 1.791, 2.33, 2.77},
00975 {4.115, 3.995, 4.345, 4.915, 5.667, 6.583, 7.547, 8.4, 8.806,
00976 8.416, 7.274, 5.963, 4.777, 3.806, 2.936, 2.114, 1.555, 1.194,
00977 0.9788, 0.8251, 0.7444, 0.7323, 0.8067, 0.4073, 0.4371, 0.8223, 1.294,
00978 1.799, 1.845, 1.715, 1.83, 2.164, 2.718, 3.019},
00979 {4.267, 4.151, 4.545, 5.138, 5.928, 6.9, 7.906, 8.818, 9.276,
00980 8.888, 7.7, 6.275, 4.923, 3.83, 2.949, 2.126, 1.57, 1.211,
00981 0.9919, 0.8349, 0.7611, 0.7613, 0.8757, 0.4158, 0.6205, 0.9285, 1.474,
00982 1.966, 1.886, 1.546, 1.534, 1.58, 1.727, 1.781},
00983 {4.357, 4.249, 4.661, 5.37, 6.245, 7.245, 8.255, 9.142, 9.591,
00984 9.187, 7.974, 6.489, 5.106, 3.983, 3.055, 2.223, 1.646, 1.263,
00985 1.021, 0.8595, 0.7835, 0.7906, 0.9308, 0.4752, 0.6857, 1.015, 1.644,
00986 2.238, 2.142, 1.682, 1.628, 1.619, 1.797, 1.869},
00987 {4.506, 4.431, 4.893, 5.628, 6.535, 7.541, 8.516, 9.354, 9.751,
00988 9.305, 8.052, 6.578, 5.209, 4.113, 3.227, 2.379, 1.745, 1.32,
00989 1.053, 0.8816, 0.796, 0.7899, 0.9061, 1.191, 0.5672, 0.9669, 1.349,
00990 1.746, 1.888, 1.592, 1.62, 1.791, 2.222, 2.485},
00991 {4.576, 4.506, 4.983, 5.745, 6.664, 7.67, 8.657, 9.446, 9.792,
00992 9.306, 8.032, 6.569, 5.251, 4.219, 3.374, 2.515, 1.835, 1.359,
00993 1.057, 0.8733, 0.7654, 0.7338, 0.8346, 1.067, 0.6038, 1.054, 1.38,
00994 1.684, 1.756, 1.589, 1.679, 2.018, 2.626, 2.986},
00995 {4.675, 4.625, 5.078, 5.828, 6.714, 7.661, 8.588, 9.4, 9.761,
00996 9.26, 7.973, 6.455, 5.133, 4.127, 3.339, 2.562, 1.884, 1.37,
00997 1.036, 0.8316, 0.7014, 0.6314, 0.6329, 0.6673, 0.7434, 1.216, 1.142,
00998 1.346, 1.515, 1.614, 1.773, 2.186, 2.758, 3.13},
00999 {4.664, 4.649, 5.144, 5.923, 6.826, 7.78, 8.69, 9.369, 9.701,
01000 9.297, 8.02, 6.479, 5.147, 4.209, 3.474, 2.764, 2.093, 1.519,
01001 1.113, 0.8486, 0.683, 0.5849, 0.5324, 0.5209, 0.5327, 1.058, 0.9241,
01002 1.048, 1.287, 1.427, 1.546, 1.782, 2.111, 2.301},
01003 {4.771, 4.822, 5.408, 6.232, 7.155, 8.095, 8.92, 9.55, 9.74,
01004 9.14, 7.776, 6.287, 5.065, 4.198, 3.518, 2.824, 2.145, 1.537,
01005 1.096, 0.8121, 0.644, 0.5446, 0.4895, 0.4683, 0.463, 0.4501, 0.773,
01006 0.7994, 0.9657, 1.138, 1.222, 1.365, 1.564, 1.666},
01007 {5.238, 5.493, 6.287, 7.287, 8.326, 9.282, 10.02, 10.41, 10.22,
01008 9.194, 7.64, 6.172, 5.058, 4.272, 3.57, 2.766, 1.962, 1.313,
01009 0.8882, 0.6342, 0.4953, 0.4159, 0.3845, 0.3668, 0.3524, 0.3383, 0.6226,
01010 0.6524, 0.702, 0.662, 0.6453, 0.6304, 0.6111, 0.5462},
01011 {5.459, 5.786, 6.717, 8.018, 9.426, 10.72, 11.67, 11.98, 11.43,
01012 9.923, 8.082, 6.51, 5.425, 4.719, 3.958, 2.937, 1.953, 1.26,
01013 0.8432, 0.5868, 0.4389, 0.3576, 0.3281, 0.3182, 0.296, 0.2263, 0.5208,
01014 0.5264, 0.5272, 0.439, 0.4076, 0.3774, 0.3392, 0.2954},
01015 {{1.93, 2.082, 2.236, 2.401, 2.486, 2.46, 2.242, 1.936, 1.632,
01016 1.309, 1.205, 0.996, 0.8843, 0.5832, 0.3788, 0.2472, 0.1935, 0.199,
01017 0.2177, 0.3668, 0.2468, 0.1727, 0.1235, 0.1211, 0.09577, 0.05738, 0.01593,
01018 0.01572, 0.01585, 0.01477, 0.01213, 0.01077, 0.008684, 0.005934},
01019 {2.406, 2.637, 3.006, 3.467, 3.941, 4.339, 4.464, 4.221, 3.692,
01020 2.961, 2.333, 1.856, 1.579, 1.321, 0.9877, 0.7954, 0.535, 0.3953,
01021 0.3269, 0.3153, 0.4016, 0.4948, 0.4946, 0.3969, 0.2986, 0.157, 0.08327,
01022 0.09294, 0.1047, 0.09143, 0.08129, 0.06982, 0.05108, 0.0324},
01023 {2.891, 3.082, 3.531, 4.114, 4.759, 5.387, 5.81, 5.856, 5.447,
01024 4.602, 3.716, 2.967, 2.422, 1.997, 1.543, 1.312, 1.043, 0.7202,
01025 0.5009, 0.3828, 0.3369, 0.3204, 0.3053, 0.2956, 0.2344, 0.3256, 0.2033,
01026 0.2183, 0.2574, 0.252, 0.2454, 0.2496, 0.2494, 0.2289},
01027 {3.257, 3.412, 3.896, 4.558, 5.307, 6.077, 6.732, 7.047, 6.849,
01028 6.037, 4.935, 3.973, 3.242, 2.755, 2.32, 1.807, 1.313, 0.9215,
01029 0.6619, 0.4999, 0.3902, 0.3134, 0.2686, 0.245, 0.2368, 0.2048, 0.1813,
01030 0.2732, 0.3298, 0.3568, 0.3526, 0.3493, 0.3549, 0.3469},
01031 {3.587, 3.682, 4.173, 4.839, 5.622, 6.472, 7.24, 7.722, 7.706,
01032 6.99, 5.826, 4.706, 3.861, 3.255, 2.675, 1.997, 1.417, 0.9866,
01033 0.7187, 0.561, 0.4546, 0.3832, 0.3527, 0.3395, 0.3279, 0.4213, 0.3649,
01034 0.4532, 0.5745, 0.6214, 0.6234, 0.6274, 0.6214, 0.5786},
01035 {3.86, 3.88, 4.313, 4.965, 5.741, 6.605, 7.439, 8.066, 8.231,
01036 7.669, 6.514, 5.29, 4.308, 3.595, 2.953, 2.22, 1.613, 1.142,
01037 0.8371, 0.6574, 0.5366, 0.4574, 0.4295, 0.4016, 0.3794, 0.5616, 0.5829,
01038 0.7168, 0.9521, 1.025, 1.053, 1.129, 1.25, 1.317},
01039 {4.081, 4.041, 4.427, 5.032, 5.788, 6.668, 7.53, 8.225, 8.513,
01040 8.083, 6.997, 5.729, 4.642, 3.826, 3.135, 2.369, 1.741, 1.266,
01041 0.9467, 0.7487, 0.6194, 0.5505, 0.6097, 0.7323, 0.5351, 0.6646, 0.7098,
01042 0.9842, 1.318, 1.49, 1.534, 1.718, 2.035, 2.285},
01043 {4.216, 4.126, 4.503, 5.084, 5.827, 6.712, 7.64, 8.42, 8.793,
01044 8.425, 7.34, 6.028, 4.838, 3.895, 3.074, 2.264, 1.681, 1.272,
01045 1.009, 0.8308, 0.7345, 0.7299, 0.8734, 1.142, 0.5401, 0.7387, 0.9717,
01046 1.328, 1.652, 1.667, 1.69, 1.954, 2.622, 3.19},
01047 {4.272, 4.151, 4.513, 5.089, 5.851, 6.773, 7.683, 8.504, 8.944,
01048 8.636, 7.572, 6.215, 4.931, 3.883, 2.974, 2.147, 1.591, 1.231,
01049 1.009, 0.859, 0.7919, 0.8024, 0.896, 0.4226, 0.5516, 0.8571, 1.354,
01050 1.883, 1.918, 1.778, 1.904, 2.264, 2.865, 3.19},
01051 {4.268, 4.149, 4.512, 5.115, 5.895, 6.826, 7.773, 8.616, 9.068,
01052 8.77, 7.691, 6.307, 4.976, 3.859, 2.932, 2.123, 1.578, 1.222,
01053 0.9995, 0.8488, 0.7895, 0.8146, 0.9335, 0.4125, 0.6071, 0.932, 1.493,
01054 1.986, 1.896, 1.55, 1.538, 1.583, 1.731, 1.792},

01055 {4.24, 4.152, 4.565, 5.21, 6.012, 6.947, 7.876, 8.687, 9.116,
01056 8.769, 7.647, 6.266, 4.953, 3.871, 2.972, 2.151, 1.594, 1.22,
01057 0.9831, 0.8291, 0.7694, 0.7899, 0.9141, 0.4431, 0.6319, 0.9495, 1.544,
01058 2.116, 2.045, 1.603, 1.547, 1.538, 1.698, 1.778},
01059 {4.22, 4.152, 4.587, 5.265, 6.061, 6.961, 7.879, 8.672, 9.045,
01060 8.614, 7.444, 6.083, 4.845, 3.844, 2.996, 2.197, 1.608, 1.21,
01061 0.9591, 0.8021, 0.7294, 0.7305, 0.8376, 1.102, 0.495, 0.808, 1.134,
01062 1.49, 1.689, 1.458, 1.475, 1.619, 1.978, 2.214},
01063 {4.139, 4.115, 4.593, 5.262, 6.061, 6.95, 7.815, 8.557, 8.84,
01064 8.311, 7.1, 5.777, 4.638, 3.742, 2.972, 2.201, 1.594, 1.167,
01065 0.9052, 0.7436, 0.6504, 0.62, 0.7047, 0.8275, 0.483, 0.7578, 0.9831,
01066 1.246, 1.432, 1.377, 1.433, 1.677, 2.116, 2.395},
01067 {4.027, 4.03, 4.521, 5.195, 5.986, 6.839, 7.642, 8.241, 8.387,
01068 7.78, 6.581, 5.331, 4.311, 3.538, 2.878, 2.173, 1.57, 1.118,
01069 0.8368, 0.6691, 0.5655, 0.5116, 0.5213, 0.5546, 0.5962, 0.7812, 0.7265,
01070 0.8823, 1.107, 1.292, 1.385, 1.633, 2.016, 2.214},
01071 {3.787, 3.852, 4.369, 5.07, 5.845, 6.625, 7.28, 7.735, 7.755,
01072 7.082, 5.9, 4.755, 3.857, 3.202, 2.64, 2.031, 1.478, 1.036,
01073 0.7546, 0.5863, 0.4836, 0.422, 0.3872, 0.3827, 0.4058, 0.6138, 0.5067,
01074 0.6048, 0.7857, 0.9735, 1.052, 1.172, 1.322, 1.358},
01075 {3.556, 3.717, 4.24, 4.935, 5.651, 6.309, 6.815, 7.1, 6.959,
01076 6.199, 5.099, 4.12, 3.39, 2.875, 2.37, 1.798, 1.291, 0.8979,
01077 0.6433, 0.4909, 0.3991, 0.3437, 0.3063, 0.2936, 0.2983, 0.283, 0.3393,
01078 0.3825, 0.4825, 0.6185, 0.6697, 0.7099, 0.7581, 0.7492},
01079 {3.344, 3.611, 4.169, 4.812, 5.45, 5.983, 6.257, 6.263, 5.844,
01080 4.989, 4.049, 3.337, 2.825, 2.403, 1.893, 1.346, 0.9248, 0.6385,
01081 0.4618, 0.3493, 0.281, 0.2397, 0.2114, 0.1989, 0.1936, 0.1739, 0.1941,
01082 0.2137, 0.2382, 0.2613, 0.2565, 0.2526, 0.2392, 0.2172},
01083 {3.617, 4.108, 4.785, 5.409, 5.873, 6.058, 5.747, 5.269, 4.577,
01084 3.776, 3.142, 2.721, 2.304, 1.764, 1.195, 0.7421, 0.4648, 0.3052,
01085 0.2132, 0.1552, 0.1201, 0.1028, 0.09496, 0.09272, 0.092, 0.06578, 0.09663,
01086 0.1039, 0.1161, 0.1127, 0.1081, 0.1019, 0.09228, 0.08144},
01087 {4.776, 5.263, 6.105, 7.209, 8.284, 9.138, 9.553, 9.22, 8.161,
01088 6.644, 5.129, 4.002, 3.207, 2.842, 2.466, 1.792, 1.171, 0.7381,
01089 0.5022, 0.4253, 0.4401, 0.5454, 0.7785, 0.6418, 0.4899, 0.3887, 0.3859,
01090 0.2463, 0.248, 0.1831, 0.1414, 0.1233, 0.104, 0.07015},
01091 {4.315, 4.602, 5.253, 6.073, 6.933, 7.732, 8.306, 8.428, 7.938,
01092 6.819, 5.489, 4.358, 3.446, 2.895, 2.476, 1.885, 1.473, 1.07,
01093 0.7373, 0.5219, 0.4618, 0.454, 0.4569, 0.4263, 0.397, 0.4683, 0.4728,
01094 0.4348, 0.4638, 0.3711, 0.3221, 0.2786, 0.2347, 0.153},
01095 {4.034, 4.197, 4.772, 5.576, 6.46, 7.358, 8.131, 8.568, 8.457,
01096 7.592, 6.31, 5.095, 4.065, 3.363, 2.711, 2.026, 1.686, 1.33,
01097 1.006, 0.7144, 0.5379, 0.4588, 0.4427, 0.451, 0.4738, 0.5938, 0.595,
01098 0.5555, 0.6354, 0.6081, 0.5877, 0.5953, 0.6217, 0.6394},
01099 {4.052, 4.154, 4.699, 5.472, 6.377, 7.353, 8.242, 8.879, 9.016,
01100 8.382, 7.119, 5.777, 4.673, 3.861, 3.197, 2.408, 1.717, 1.19,
01101 0.8643, 0.6598, 0.5193, 0.4314, 0.3937, 0.3863, 0.4011, 0.3591, 0.4105,
01102 0.4673, 0.5539, 0.5717, 0.5531, 0.5518, 0.5563, 0.5379},
01103 {4.153, 4.213, 4.735, 5.466, 6.334, 7.3, 8.25, 8.998, 9.286,
01104 8.779, 7.558, 6.165, 5, 4.154, 3.419, 2.575, 1.849, 1.298,
01105 0.9519, 0.7439, 0.6103, 0.537, 0.545, 0.5816, 0.5683, 0.7049, 0.6064,
01106 0.7504, 0.9181, 0.9021, 0.8857, 0.8707, 0.8466, 0.7728},
01107 {4.248, 4.254, 4.738, 5.442, 6.269, 7.19, 8.14, 8.939, 9.34,
01108 8.978, 7.82, 6.411, 5.184, 4.273, 3.525, 2.664, 1.942, 1.394,
01109 1.033, 0.8143, 0.6732, 0.5954, 0.6154, 0.6352, 0.7198, 0.8339, 0.8119,
01110 1.046, 1.311, 1.244, 1.264, 1.359, 1.508, 1.629},
01111 {4.354, 4.315, 4.761, 5.42, 6.219, 7.126, 8.036, 8.844, 9.254,
01112 8.926, 7.833, 6.444, 5.174, 4.22, 3.457, 2.639, 1.948, 1.432,
01113 1.087, 0.8685, 0.7349, 0.6908, 0.8, 0.9447, 0.6798, 0.8977, 1.108,
01114 1.425, 1.666, 1.47, 1.449, 1.545, 1.768, 1.943},
01115 {4.362, 4.274, 4.672, 5.312, 6.096, 7.012, 7.964, 8.785, 9.19,
01116 8.837, 7.774, 6.356, 5.026, 3.988, 3.137, 2.353, 1.747, 1.332,
01117 1.065, 0.8964, 0.8178, 0.823, 0.9551, 1.158, 0.5854, 0.9884, 1.483,
01118 2.035, 2.091, 1.603, 1.521, 1.55, 1.723, 1.854},
01119 {4.3, 4.174, 4.537, 5.167, 5.965, 6.89, 7.821, 8.663, 9.083,
01120 8.735, 7.634, 6.25, 4.936, 3.878, 2.975, 2.165, 1.611, 1.244,
01121 1.01, 0.864, 0.8216, 0.8541, 0.947, 0.4242, 0.4811, 0.9771, 1.569,
01122 2.131, 2.042, 1.616, 1.566, 1.597, 1.77, 1.859},
01123 {4.191, 4.082, 4.474, 5.086, 5.853, 6.754, 7.678, 8.491, 8.887,
01124 8.503, 7.408, 6.052, 4.775, 3.739, 2.846, 2.066, 1.531, 1.182,
01125 0.9574, 0.8217, 0.7847, 0.8129, 0.9004, 0.3845, 0.5912, 0.9488, 1.54,
01126 2.073, 1.987, 1.591, 1.544, 1.55, 1.68, 1.746},
01127 {4.067, 4, 4.384, 4.981, 5.736, 6.618, 7.486, 8.26, 8.602,
01128 8.169, 7.063, 5.748, 4.553, 3.579, 2.743, 1.997, 1.466, 1.117,
01129 0.8947, 0.756, 0.7085, 0.7246, 0.8226, 0.3804, 0.4137, 0.8378, 1.338,
01130 1.831, 1.823, 1.452, 1.396, 1.406, 1.546, 1.607},
01131 {3.852, 3.824, 4.228, 4.825, 5.559, 6.404, 7.233, 7.922, 8.179,
01132 7.678, 6.56, 5.348, 4.269, 3.383, 2.62, 1.89, 1.369, 1.029,
01133 0.8128, 0.6781, 0.613, 0.6018, 0.6705, 0.6956, 0.4126, 0.6338, 0.8946,
01134 1.186, 1.405, 1.278, 1.277, 1.398, 1.661, 1.83},
01135 {3.577, 3.6, 4.019, 4.632, 5.336, 6.114, 6.899, 7.485, 7.599,
01136 6.995, 5.898, 4.813, 3.907, 3.169, 2.502, 1.82, 1.298, 0.9403,
01137 0.7213, 0.5834, 0.4987, 0.4571, 0.4767, 0.4929, 0.357, 0.5342, 0.6218,
01138 0.8321, 1.072, 1.133, 1.182, 1.326, 1.548, 1.687},
01139 {3.251, 3.338, 3.79, 4.396, 5.094, 5.833, 6.465, 6.856, 6.763,
01140 6.045, 5.018, 4.11, 3.394, 2.814, 2.265, 1.671, 1.175, 0.8195,
01141 0.6074, 0.4777, 0.3953, 0.3477, 0.3295, 0.3281, 0.3202, 0.3996, 0.4085,


```

01142     0.5264, 0.6945, 0.8446, 0.9062, 1.002, 1.145, 1.182},
01143 {2.844, 3.014, 3.474, 4.053, 4.689, 5.317, 5.776, 5.907, 5.606,
01144 4.864, 3.98, 3.263, 2.726, 2.292, 1.824, 1.308, 0.8985, 0.62,
01145 0.4532, 0.3508, 0.2879, 0.2496, 0.2268, 0.217, 0.2089, 0.1723, 0.2384,
01146 0.3033, 0.3816, 0.457, 0.4738, 0.4945, 0.522, 0.511},
01147 {2.32, 2.558, 2.981, 3.486, 3.981, 4.383, 4.505, 4.35, 3.92,
01148 3.298, 2.688, 2.238, 1.901, 1.574, 1.197, 0.8142, 0.5528, 0.3893,
01149 0.2874, 0.2207, 0.179, 0.153, 0.1353, 0.126, 0.118, 0.0996, 0.1038,
01150 0.1254, 0.1556, 0.1673, 0.1637, 0.1606, 0.1599, 0.1512},
01151 {1.604, 1.812, 2.085, 2.33, 2.517, 2.565, 2.341, 2.07, 1.768,
01152 1.479, 1.254, 1.072, 0.8657, 0.6304, 0.4269, 0.2773, 0.1887, 0.1365,
01153 0.104, 0.08106, 0.06679, 0.05742, 0.0504, 0.04538, 0.04064, 0.03211,
01154 0.02611,
01155 0.02727, 0.03066, 0.03125, 0.02913, 0.02667, 0.02485, 0.02217},
01156 {0.4429, 0.4652, 0.4579, 0.4357, 0.388, 0.3401, 0.2901, 0.2551, 0.2192,
01157 0.1788, 0.1348, 0.0914, 0.05502, 0.03148, 0.01858, 0.01216, 0.009078,
01158 0.007534,
01159 0.006492, 0.006257, 0.006301, 0.006115, 0.005378, 0.005084, 0.005022,
01160 0.002766, 0.00246,
01161 0.001431, 0.001208, 0.0014, 0.001257, 0.001091, 0.0009076, 0.0005632}},
01162 {{6.159, 6.267, 7.137, 8.441, 9.868, 11.32, 12.68, 13.44, 13.31,
01163 11.97, 9.899, 7.982, 6.529, 5.456, 4.582, 3.411, 2.447, 1.735,
01164 1.201, 0.8902, 0.7385, 0.82, 0.8247, 0.6792, 0.5978, 0.7594, 1.311,
01165 0.8001, 0.7222, 0.5196, 0.4, 0.3513, 0.3049, 0.2033},
01166 {6.025, 6.218, 7.108, 8.305, 9.599, 10.86, 11.89, 12.5, 12.4,
01167 11.24, 9.372, 7.562, 6.173, 5.198, 4.359, 3.187, 2.293, 1.674,
01168 1.209, 0.8758, 0.6848, 0.7086, 0.7544, 0.6892, 0.6462, 0.8934, 1.25,
01169 0.9268, 0.9078, 0.7087, 0.6172, 0.5399, 0.4671, 0.304},
01170 {5.423, 5.508, 6.112, 6.997, 8.017, 9.107, 10.07, 10.75, 10.95,
01171 10.26, 8.775, 7.139, 5.746, 4.723, 3.902, 2.898, 2.14, 1.579,
01172 1.155, 0.86, 0.6747, 0.5805, 0.5689, 0.5958, 0.6918, 0.8863, 1.028,
01173 0.8369, 0.8899, 0.8223, 0.7867, 0.7493, 0.7297, 0.6911},
01174 {5.03, 5.007, 5.533, 6.337, 7.269, 8.306, 9.353, 10.16, 10.53,
01175 10.09, 8.802, 7.176, 5.772, 4.785, 3.999, 3.116, 2.292, 1.613,
01176 1.161, 0.8825, 0.7044, 0.602, 0.5675, 0.5799, 0.633, 0.5335, 0.75,
01177 0.782, 0.8887, 0.8566, 0.8053, 0.779, 0.7512, 0.6894},
01178 {4.854, 4.795, 5.239, 5.991, 6.89, 7.916, 8.949, 9.794, 10.25,
01179 9.932, 8.787, 7.214, 5.806, 4.824, 4.046, 3.151, 2.318, 1.656,
01180 1.216, 0.9337, 0.7505, 0.6429, 0.6093, 0.6241, 0.6313, 0.828, 0.8414,
01181 1.007, 1.191, 1.083, 1.037, 1.002, 0.963, 0.8665},
01182 {4.689, 4.616, 5.08, 5.801, 6.661, 7.626, 8.619, 9.471, 9.972,
01183 9.711, 8.582, 7.039, 5.65, 4.664, 3.923, 3.04, 2.236, 1.63,
01184 1.223, 0.9546, 0.7831, 0.6819, 0.6534, 0.6599, 0.5823, 0.9021, 1.016,
01185 1.267, 1.495, 1.395, 1.398, 1.508, 1.711, 1.859},
01186 {4.575, 4.466, 4.925, 5.634, 6.479, 7.412, 8.343, 9.218, 9.74,
01187 9.468, 8.319, 6.801, 5.4, 4.37, 3.571, 2.713, 2.014, 1.498,
01188 1.158, 0.9364, 0.7954, 0.7273, 0.7681, 0.8566, 0.6687, 0.9463, 1.219,
01189 1.575, 1.788, 1.546, 1.525, 1.635, 1.887, 2.09},
01190 {4.408, 4.299, 4.735, 5.433, 6.275, 7.22, 8.175, 9.032, 9.47,
01191 9.097, 7.942, 6.45, 5.08, 4.019, 3.16, 2.338, 1.735, 1.324,
01192 1.063, 0.9095, 0.8421, 0.8249, 0.8269, 0.6041, 0.6079, 1.07, 1.601,
01193 2.169, 2.204, 1.68, 1.593, 1.624, 1.805, 1.939},
01194 {4.26, 4.136, 4.516, 5.172, 5.992, 6.946, 7.925, 8.739, 9.102,
01195 8.667, 7.534, 6.13, 4.865, 3.837, 2.924, 2.126, 1.572, 1.207,
01196 0.9693, 0.8391, 0.8016, 0.8213, 0.8584, 0.4114, 0.632, 0.9986, 1.59,
01197 2.132, 2.045, 1.617, 1.567, 1.595, 1.754, 1.846},
01198 {4.068, 3.944, 4.321, 4.953, 5.759, 6.696, 7.598, 8.349, 8.636,
01199 8.134, 6.966, 5.681, 4.531, 3.592, 2.726, 1.981, 1.454, 1.112,
01200 0.8863, 0.7642, 0.7336, 0.7503, 0.7883, 0.3649, 0.5742, 0.9245, 1.482,
01201 1.979, 1.91, 1.537, 1.492, 1.497, 1.609, 1.655},
01202 {3.784, 3.702, 4.08, 4.686, 5.436, 6.274, 7.107, 7.772, 7.978,
01203 7.457, 6.38, 5.207, 4.171, 3.317, 2.572, 1.856, 1.347, 1.012,
01204 0.8031, 0.6877, 0.6459, 0.6441, 0.6934, 0.3392, 0.5146, 0.7711, 1.206,
01205 1.627, 1.668, 1.358, 1.31, 1.315, 1.411, 1.432},
01206 {3.39, 3.409, 3.832, 4.448, 5.159, 5.932, 6.66, 7.149, 7.206,
01207 6.618, 5.602, 4.608, 3.743, 3.01, 2.347, 1.689, 1.206, 0.8923,
01208 0.6966, 0.5763, 0.5136, 0.4878, 0.5216, 0.5783, 0.3499, 0.515, 0.7012,
01209 0.9131, 1.167, 1.133, 1.139, 1.212, 1.359, 1.445},
01210 {3.031, 3.122, 3.551, 4.115, 4.781, 5.496, 6.101, 6.433, 6.32,
01211 5.654, 4.707, 3.886, 3.211, 2.629, 2.053, 1.473, 1.024, 0.7318,
01212 0.5579, 0.445, 0.3748, 0.3356, 0.3272, 0.3261, 0.3502, 0.4067, 0.4482,
01213 0.5625, 0.7534, 0.8328, 0.8615, 0.9261, 1.038, 1.075},
01214 {2.556, 2.697, 3.11, 3.64, 4.251, 4.887, 5.363, 5.492, 5.176,
01215 4.453, 3.662, 3.064, 2.599, 2.164, 1.677, 1.161, 0.7816, 0.5445,
01216 0.4076, 0.3171, 0.258, 0.2227, 0.2043, 0.1946, 0.1903, 0.2423, 0.2411,
01217 0.2984, 0.3661, 0.4305, 0.4483, 0.4735, 0.5096, 0.5082},
01218 {1.982, 2.163, 2.522, 2.962, 3.444, 3.894, 4.12, 3.996, 3.538,
01219 2.915, 2.39, 2.044, 1.761, 1.418, 1.026, 0.6684, 0.4452, 0.3147,
01220 0.2354, 0.1814, 0.1474, 0.1272, 0.1136, 0.1042, 0.09334, 0.07244, 0.09453,
01221 0.1067, 0.1323, 0.1309, 0.1255, 0.1235, 0.1251, 0.1207},
01222 {1.313, 1.48, 1.706, 1.932, 2.113, 2.193, 2.081, 1.804, 1.487,
01223 1.196, 0.9808, 0.8365, 0.6791, 0.4931, 0.3304, 0.2112, 0.1439, 0.1054,
01224 0.08052, 0.06314, 0.05248, 0.04667, 0.0419, 0.03731, 0.03192, 0.02135,
01225 0.01682,
01226 0.0156, 0.01767, 0.01723, 0.0161, 0.01526, 0.0148, 0.01411},
01227 {0.242, 0.2311, 0.2162, 0.1962, 0.1752, 0.1604, 0.1387, 0.1112, 0.08183,
01228 0.05815, 0.04045, 0.02676, 0.01677, 0.01075, 0.007653, 0.005984, 0.00512,

```

```
01229      0.004795,
01230      0.004786, 0.004999, 0.004952, 0.004352, 0.003443, 0.002664, 0.002223,
01231      0.001163, 0.001542,
01232      0.0002821, 0.0001951, 0.000206, 0.0001656, 0.0001206, 8.303e-05,
01233      5.901e-05},
01234      {0.0001232, 0.0001559, 0.0001539, 0.0001693, 0.0002134, 0.0002031,
01235      0.0001037, 1.126e-05, 5.382e-06,
01236      1.867e-06, 5.983e-07, 2.464e-07, 1.576e-07, 1.322e-07, 1.312e-07,
01237      1.319e-07, 3.921e-07, 3.583e-06,
01238      3.815e-05, 6.754e-05, 0.0001004, 0.0002135, 0.0004217, 0.0007681,
01239      0.001524, 0.0004274, 0.000876,
01240      2.698e-05, 1.328e-12, 1.445e-13, 9.798e-14, 8.583e-14, 9.786e-14,
01241      1.774e-13}},
01242      {{6.595, 6.532, 7.313, 8.453, 9.864, 11.47, 13.06, 14.28, 14.67,
01243      13.68, 11.56, 9.275, 7.452, 6.201, 5.275, 4.16, 2.898, 2.003,
01244      1.4, 1.04, 0.7754, 0.7071, 0.7598, 0.799, 0.825, 0.9217, 1.851,
01245      1.254, 1.138, 0.8159, 0.6311, 0.5427, 0.4614, 0.3814}},
01246      {6.516, 6.556, 7.327, 8.526, 9.924, 11.42, 12.85, 13.82, 14.03,
01247      13.05, 11.03, 8.863, 7.108, 5.878, 4.956, 3.797, 2.704, 1.92,
01248      1.344, 0.9685, 0.7276, 0.6364, 0.6746, 0.7239, 0.786, 0.9333, 1.793,
01249      1.344, 1.234, 0.8885, 0.7949, 0.6932, 0.5878, 0.4871}},
01250      {6.179, 6.202, 6.853, 7.807, 8.924, 10.13, 11.21, 12.01, 12.29,
01251      11.63, 10.05, 8.152, 6.536, 5.386, 4.503, 3.473, 2.521, 1.809,
01252      1.273, 0.9058, 0.6837, 0.5774, 0.5746, 0.6269, 0.7726, 0.9434, 1.275,
01253      1.102, 1.148, 0.9922, 0.9195, 0.8713, 0.8162, 0.7358}},
01254      {5.401, 5.302, 5.812, 6.634, 7.64, 8.785, 9.902, 10.82, 11.3,
01255      10.96, 9.696, 7.981, 6.412, 5.281, 4.41, 3.469, 2.606, 1.892,
01256      1.37, 1.034, 0.8087, 0.6766, 0.6565, 0.6981, 0.7901, 0.6904, 1.01,
01257      1.062, 1.192, 1.063, 1.016, 0.9639, 0.8911, 0.7914}},
01258      {5.101, 4.973, 5.426, 6.18, 7.138, 8.24, 9.32, 10.29, 10.9,
01259      10.75, 9.665, 8.035, 6.469, 5.319, 4.452, 3.502, 2.649, 1.941,
01260      1.431, 1.09, 0.869, 0.7456, 0.7339, 0.7833, 0.8079, 1.059, 1.104,
01261      1.303, 1.515, 1.253, 1.185, 1.131, 1.076, 0.9437}},
01262      {4.936, 4.795, 5.272, 5.985, 6.878, 7.91, 8.989, 9.922, 10.53,
01263      10.37, 9.278, 7.698, 6.176, 5.044, 4.178, 3.263, 2.472, 1.849,
01264      1.402, 1.087, 0.8859, 0.7846, 0.8226, 0.8854, 0.9635, 1.037, 1.251,
01265      1.527, 1.706, 1.5, 1.503, 1.644, 1.914, 2.113}},
01266      {4.796, 4.617, 5.024, 5.703, 6.591, 7.617, 8.632, 9.544, 10.07,
01267      9.749, 8.552, 6.983, 5.55, 4.462, 3.573, 2.707, 2.021, 1.537,
01268      1.216, 1.017, 0.9039, 0.8702, 0.9836, 1.21, 0.6125, 1.009, 1.311,
01269      1.688, 1.862, 1.575, 1.568, 1.696, 2.001, 2.214}},
01270      {4.522, 4.356, 4.742, 5.465, 6.36, 7.357, 8.359, 9.269, 9.706,
01271      9.237, 7.95, 6.476, 5.137, 4.086, 3.214, 2.373, 1.75, 1.33,
01272      1.071, 0.9379, 0.8929, 0.9071, 0.9736, 1.305, 0.5218, 1.054, 1.605,
01273      2.105, 1.976, 1.563, 1.521, 1.56, 1.765, 1.875}},
01274      {4.201, 4.084, 4.453, 5.134, 5.998, 7.007, 8.042, 8.894, 9.218,
01275      8.665, 7.393, 5.966, 4.728, 3.77, 2.956, 2.16, 1.585, 1.199,
01276      0.9637, 0.8579, 0.8414, 0.8686, 0.8189, 1.154, 0.4693, 0.9934, 1.568,
01277      2.075, 1.962, 1.563, 1.524, 1.545, 1.704, 1.786}},
01278      {3.87, 3.761, 4.135, 4.74, 5.547, 6.523, 7.533, 8.287, 8.542,
01279      7.978, 6.743, 5.463, 4.36, 3.491, 2.739, 1.993, 1.453, 1.095,
01280      0.8767, 0.7822, 0.7664, 0.777, 0.8145, 1.109, 0.4094, 0.8854, 1.413,
01281      1.91, 1.872, 1.47, 1.421, 1.428, 1.538, 1.583}},
01282      {3.565, 3.517, 3.908, 4.525, 5.299, 6.159, 6.982, 7.581, 7.734,
01283      7.15, 6.028, 4.918, 3.993, 3.242, 2.541, 1.833, 1.321, 0.9862,
01284      0.7851, 0.6877, 0.6504, 0.6409, 0.6657, 0.7916, 0.3852, 0.627, 0.8774,
01285      1.306, 1.713, 1.397, 1.317, 1.308, 1.379, 1.377}},
01286      {3.27, 3.307, 3.718, 4.324, 5.008, 5.72, 6.391, 6.82, 6.844,
01287      6.25, 5.256, 4.321, 3.562, 2.929, 2.309, 1.67, 1.183, 0.8581,
01288      0.6613, 0.5437, 0.4817, 0.4549, 0.4828, 0.4971, 0.343, 0.4517, 0.5928,
01289      0.7482, 1.114, 1.156, 1.127, 1.142, 1.266, 1.325}},
01290      {2.881, 2.972, 3.365, 3.885, 4.479, 5.095, 5.612, 5.869, 5.739,
01291      5.109, 4.233, 3.497, 2.928, 2.45, 1.923, 1.37, 0.937, 0.6588,
01292      0.4974, 0.3913, 0.3216, 0.2799, 0.263, 0.2476, 0.2702, 0.3664, 0.3897,
01293      0.4754, 0.6181, 0.6968, 0.7144, 0.7507, 0.8199, 0.8256}},
01294      {2.352, 2.522, 2.914, 3.377, 3.888, 4.391, 4.73, 4.773, 4.456,
01295      3.814, 3.103, 2.576, 2.19, 1.824, 1.372, 0.9129, 0.606, 0.4281,
01296      0.3241, 0.25, 0.1992, 0.1685, 0.1489, 0.1316, 0.116, 0.1598, 0.1448,
01297      0.1805, 0.2224, 0.2379, 0.2369, 0.2454, 0.2679, 0.2718}},
01298      {1.666, 1.833, 2.135, 2.486, 2.847, 3.14, 3.202, 3.006, 2.612,
01299      2.127, 1.726, 1.486, 1.277, 0.9733, 0.6654, 0.4233, 0.2852, 0.2051,
01300      0.1537, 0.1174, 0.09422, 0.08017, 0.06975, 0.06009, 0.04775, 0.03319,
01301      0.03371,
01302      0.03896, 0.04544, 0.04203, 0.03927, 0.03814, 0.03917, 0.04012}},
01303      {0.8975, 0.9719, 1.03, 1.066, 1.034, 0.9374, 0.7957, 0.662, 0.5656,
01304      0.4856, 0.4141, 0.3239, 0.2283, 0.1478, 0.09439, 0.06056, 0.04188,
01305      0.03179,
01306      0.02625, 0.02293, 0.02134, 0.01999, 0.01796, 0.01508, 0.01136, 0.006243,
01307      0.005399,
01308      0.002554, 0.002671, 0.002877, 0.002693, 0.002456, 0.002169, 0.001592}},
01309      {0.005568, 0.003081, 0.001936, 0.001388, 0.001138, 0.0009141, 0.0003913,
01310      7.042e-05, 1.305e-05,
01311      9.014e-06, 5.819e-06, 3.047e-06, 1.303e-06, 5.602e-07, 2.183e-07,
01312      1.757e-07, 3.825e-07, 2.566e-06,
01313      1.334e-05, 1.436e-05, 1.976e-05, 7.261e-05, 0.0002657, 0.0005962,
01314      0.001653, 0.0002773, 0.0008521,
01315      1.309e-06, 3.72e-14, 2.315e-16, 2.404e-15, 7.283e-17, 5.816e-17,
```

```

01316     1.165e-16},
01317     {5.606e-05, 7.174e-05, 7.065e-05, 8.779e-05, 0.0001175, 0.0001418,
01318     6.181e-05, 7.462e-06, 8.135e-06,
01319     6.922e-06, 3.21e-06, 1.063e-06, 3.185e-07, 7.307e-08, 1.298e-08,
01320     1.751e-08, 6.792e-08, 5.277e-07,
01321     7.612e-06, 1.832e-05, 4.78e-05, 0.0001019, 0.0001703, 0.0003801, 0.001213,
01322     0.0002105, 0.0006011,
01323     2.875e-06, 7.798e-13, 1.214e-13, 8.329e-14, 7.553e-14, 1.014e-13,
01324     1.901e-13}}
01325 };
01326
01327 #ifdef _OPENACC
01328 #pragma acc declare copyin(clim_oh_var)
01329 #endif
01330
01331 double clim_oh(
01332     double t,
01333     double lat,
01334     double p) {
01335
01336     /* Get seconds since begin of year... */
01337     double sec = FMOD(t, 365.25 * 86400.);
01338     while (sec < 0)
01339         sec += 365.25 * 86400.;
01340
01341     /* Check pressure... */
01342     if (p < clim_oh_ps[0])
01343         p = clim_oh_ps[0];
01344     else if (p > clim_oh_ps[33])
01345         p = clim_oh_ps[33];
01346
01347     /* Check latitude... */
01348     if (lat < clim_oh_lats[0])
01349         lat = clim_oh_lats[0];
01350     else if (lat > clim_oh_lats[17])
01351         lat = clim_oh_lats[17];
01352
01353     /* Get indices... */
01354     int isec = locate_irr(clim_oh_secs, 12, sec);
01355     int ilat = locate_reg(clim_oh_lats, 18, lat);
01356     int ip = locate_irr(clim_oh_ps, 34, p);
01357
01358     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01359     double aux00 = LIN(clim_oh_ps[ip],
01360                       clim_oh_var[isec][ilat][ip],
01361                       clim_oh_ps[ip + 1],
01362                       clim_oh_var[isec][ilat][ip + 1], p);
01363     double aux01 = LIN(clim_oh_ps[ip],
01364                       clim_oh_var[isec][ilat + 1][ip],
01365                       clim_oh_ps[ip + 1],
01366                       clim_oh_var[isec][ilat + 1][ip + 1], p);
01367     double aux10 = LIN(clim_oh_ps[ip],
01368                       clim_oh_var[isec + 1][ilat][ip],
01369                       clim_oh_ps[ip + 1],
01370                       clim_oh_var[isec + 1][ilat][ip + 1], p);
01371     double aux11 = LIN(clim_oh_ps[ip],
01372                       clim_oh_var[isec + 1][ilat + 1][ip],
01373                       clim_oh_ps[ip + 1],
01374                       clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01375     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01376     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01377     aux00 = LIN(clim_oh_secs[isec], aux00, clim_oh_secs[isec + 1], aux11, sec);
01378     return GSL_MAX(1e6 * aux00, 0.0);
01379 }
01380
01381 /*****
01382
01383 static double clim_tropo_secs[12] = {
01384     1209600.00, 3888000.00, 6393600.00,
01385     9072000.00, 11664000.00, 14342400.00,
01386     16934400.00, 19612800.00, 22291200.00,
01387     24883200.00, 27561600.00, 30153600.00
01388 };
01389
01390 #ifdef _OPENACC
01391 #pragma acc declare copyin(clim_tropo_secs)
01392 #endif
01393
01394 static double clim_tropo_lats[73]
01395 = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01396     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01397     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01398     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01399     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01400     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01401     75, 77.5, 80, 82.5, 85, 87.5, 90
01402 };

```

```
01403
01404 #ifndef _OPENACC
01405 #pragma acc declare copyin(clim_tropo_lats)
01406 #endif
01407
01408 static double clim_tropo_tps[12][73]
01409 = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01410     297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01411     175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01412     99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01413     98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01414     152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01415     277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01416     275.3, 275.6, 275.4, 274.1, 273.5},
01417 {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01418     300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01419     150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01420     98.88, 98.35, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01421     98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01422     220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01423     284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01424     287.5, 286.2, 285.8},
01425 {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01426     297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01427     161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01428     100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01429     99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01430     186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01431     279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01432     304.3, 304.9, 306, 306.6, 306.2, 306},
01433 {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01434     290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01435     195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01436     102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01437     99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01438     148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01439     263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01440     315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01441 {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01442     260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01443     205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01444     101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01445     102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01446     165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01447     273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01448     325.3, 325.8, 325.8},
01449 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01450     222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01451     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01452     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01453     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01454     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01455     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01456     308.5, 312.2, 313.1, 313.3},
01457 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01458     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01459     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01460     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01461     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01462     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01463     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01464     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01465 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01466     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01467     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01468     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01469     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01470     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01471     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01472     278.2, 282.6, 287.4, 290.9, 292.5, 293},
01473 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01474     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01475     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01476     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01477     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01478     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01479     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01480     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01481 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01482     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01483     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01484     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01485     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01486     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01487     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01488     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01489     305.1},
```

```

01490 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01491 253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01492 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01493 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01494 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01495 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01496 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01497 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01498 {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01499 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01500 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01501 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01502 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01503 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01504 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01505 281.7, 281.1, 281.2}
01506 };
01507
01508 #ifdef _OPENACC
01509 #pragma acc declare copyin(clim_tropo_tps)
01510 #endif
01511
01512 double clim_tropo(
01513     double t,
01514     double lat) {
01515
01516     /* Get seconds since begin of year... */
01517     double sec = FMOD(t, 365.25 * 86400.);
01518     while (sec < 0)
01519         sec += 365.25 * 86400.;
01520
01521     /* Get indices... */
01522     int isec = locate_irr(clim_tropo_secs, 12, sec);
01523     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01524
01525     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01526     double p0 = LIN(clim_tropo_lats[ilat],
01527                     clim_tropo_tps[isec][ilat],
01528                     clim_tropo_lats[ilat + 1],
01529                     clim_tropo_tps[isec][ilat + 1], lat);
01530     double p1 = LIN(clim_tropo_lats[ilat],
01531                     clim_tropo_tps[isec + 1][ilat],
01532                     clim_tropo_lats[ilat + 1],
01533                     clim_tropo_tps[isec + 1][ilat + 1], lat);
01534     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01535 }
01536
01537 /*****
01538
01539 void day2doy(
01540     int year,
01541     int mon,
01542     int day,
01543     int *doy) {
01544
01545     const int
01546         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01547         d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01548
01549     /* Get day of year... */
01550     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01551         *doy = d01[mon - 1] + day - 1;
01552     else
01553         *doy = d0[mon - 1] + day - 1;
01554 }
01555
01556 /*****
01557
01558 void doy2day(
01559     int year,
01560     int doy,
01561     int *mon,
01562     int *day) {
01563
01564     const int
01565         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01566         d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01567
01568     int i;
01569
01570     /* Get month and day... */
01571     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01572         for (i = 11; i > 0; i--)
01573             if (d01[i] <= doy)
01574                 break;
01575         *mon = i + 1;
01576         *day = doy - d01[i] + 1;

```

```

01577     } else {
01578         for (i = 11; i > 0; i--)
01579             if (d0[i] <= doy)
01580                 break;
01581         *mon = i + 1;
01582         *day = doy - d0[i] + 1;
01583     }
01584 }
01585
01586 /*****
01587
01588 void geo2cart(
01589     double z,
01590     double lon,
01591     double lat,
01592     double *x) {
01593
01594     double radius = z + RE;
01595     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01596     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01597     x[2] = radius * sin(lat / 180. * M_PI);
01598 }
01599
01600 /*****
01601
01602 void get_met(
01603     ctl_t *ctl,
01604     double t,
01605     met_t **met0,
01606     met_t **met1) {
01607
01608     static int init;
01609
01610     met_t *mets;
01611
01612     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01613
01614     /* Set timer... */
01615     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01616
01617     /* Init... */
01618     if (t == ctl->t_start || !init) {
01619         init = 1;
01620
01621         /* Read meteo data... */
01622         get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01623         if (!read_met(ctl, filename, *met0))
01624             ERRMSG("Cannot open file!");
01625
01626         get_met_help(t + 1.0 * ctl->direction, 1, ctl->metbase, ctl->dt_met,
01627             filename);
01628         if (!read_met(ctl, filename, *met1))
01629             ERRMSG("Cannot open file!");
01630
01631         /* Update GPU... */
01632 #ifdef _OPENACC
01633         met_t *met0up = *met0;
01634         met_t *met1up = *met1;
01635         #pragma acc update device(met0up[:1], met1up[:1])
01636 #endif
01637
01638         /* Caching... */
01639         if (ctl->met_cache && t != ctl->t_stop) {
01640             get_met_help(t + 1.1 * ctl->dt_met * ctl->direction, ctl->direction,
01641                 ctl->metbase, ctl->dt_met, cachefile);
01642             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01643             LOG(1, "Caching: %s", cachefile);
01644             if (system(cmd) != 0)
01645                 WARN("Caching command failed!");
01646         }
01647     }
01648
01649     /* Read new data for forward trajectories... */
01650     if (t > (*met1)->time) {
01651
01652         /* Pointer swap... */
01653         mets = *met1;
01654         *met1 = *met0;
01655         *met0 = mets;
01656
01657         /* Read new meteo data... */
01658         get_met_help(t, 1, ctl->metbase, ctl->dt_met, filename);
01659         if (!read_met(ctl, filename, *met1))
01660             ERRMSG("Cannot open file!");
01661
01662         /* Update GPU... */
01663 #ifdef _OPENACC

```

```

01664     met_t *metlup = *metl;
01665 #pragma acc update device(metlup[:1])
01666 #endif
01667
01668     /* Caching... */
01669     if (ctl->met_cache && t != ctl->t_stop) {
01670         get_met_help(t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met, cachefile);
01671         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01672         LOG(1, "Caching: %s", cachefile);
01673         if (system(cmd) != 0)
01674             WARN("Caching command failed!");
01675     }
01676 }
01677
01678 /* Read new data for backward trajectories... */
01679 if (t < (*met0)->time) {
01680
01681     /* Pointer swap... */
01682     mets = *metl;
01683     *metl = *met0;
01684     *met0 = mets;
01685
01686     /* Read new meteo data... */
01687     get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01688     if (!read_met(ctl, filename, *met0))
01689         ERRMSG("Cannot open file!");
01690
01691     /* Update GPU... */
01692 #ifdef _OPENACC
01693     met_t *met0up = *met0;
01694 #pragma acc update device(met0up[:1])
01695 #endif
01696
01697     /* Caching... */
01698     if (ctl->met_cache && t != ctl->t_stop) {
01699         get_met_help(t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met, cachefile);
01700         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01701         LOG(1, "Caching: %s", cachefile);
01702         if (system(cmd) != 0)
01703             WARN("Caching command failed!");
01704     }
01705 }
01706
01707 /* Check that grids are consistent... */
01708 if ((*met0)->nx != (*met1)->nx
01709     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01710     ERRMSG("Meteo grid dimensions do not match!");
01711 for (int ix = 0; ix < (*met0)->nx; ix++)
01712     if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01713         ERRMSG("Meteo grid longitudes do not match!");
01714 for (int iy = 0; iy < (*met0)->ny; iy++)
01715     if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01716         ERRMSG("Meteo grid latitudes do not match!");
01717 for (int ip = 0; ip < (*met0)->np; ip++)
01718     if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01719         ERRMSG("Meteo grid pressure levels do not match!");
01720 }
01721
01722 /*****
01723
01724 void get_met_help(
01725     double t,
01726     int direct,
01727     char *metbase,
01728     double dt_met,
01729     char *filename) {
01730
01731     char repl[LEN];
01732
01733     double t6, r;
01734
01735     int year, mon, day, hour, min, sec;
01736
01737     /* Round time to fixed intervals... */
01738     if (direct == -1)
01739         t6 = floor(t / dt_met) * dt_met;
01740     else
01741         t6 = ceil(t / dt_met) * dt_met;
01742
01743     /* Decode time... */
01744     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01745
01746     /* Set filename... */
01747     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01748     sprintf(repl, "%d", year);
01749     get_met_replace(filename, "YYYY", repl);
01750     sprintf(repl, "%02d", mon);

```

```

01751 get_met_replace(filename, "MM", repl);
01752 sprintf(repl, "%02d", day);
01753 get_met_replace(filename, "DD", repl);
01754 sprintf(repl, "%02d", hour);
01755 get_met_replace(filename, "HH", repl);
01756 }
01757
01758 /*****
01759
01760 void get_met_replace(
01761     char *orig,
01762     char *search,
01763     char *repl) {
01764
01765     char buffer[LEN], *ch;
01766
01767     /* Iterate... */
01768     for (int i = 0; i < 3; i++) {
01769
01770         /* Replace sub-string... */
01771         if (!(ch = strstr(orig, search)))
01772             return;
01773         strncpy(buffer, orig, (size_t) (ch - orig));
01774         buffer[ch - orig] = 0;
01775         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01776         orig[0] = 0;
01777         strcpy(orig, buffer);
01778     }
01779 }
01780
01781 /*****
01782
01783 void intpol_met_space_3d(
01784     met_t * met,
01785     float array[EX][EY][EP],
01786     double p,
01787     double lon,
01788     double lat,
01789     double *var,
01790     int *ci,
01791     double *cw,
01792     int init) {
01793
01794     /* Initialize interpolation... */
01795     if (init) {
01796
01797         /* Check longitude... */
01798         if (met->lon[met->nx - 1] > 180 && lon < 0)
01799             lon += 360;
01800
01801         /* Get interpolation indices... */
01802         ci[0] = locate_irr(met->p, met->np, p);
01803         ci[1] = locate_reg(met->lon, met->nx, lon);
01804         ci[2] = locate_reg(met->lat, met->ny, lat);
01805
01806         /* Get interpolation weights... */
01807         cw[0] = (met->p[ci[0] + 1] - p)
01808             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01809         cw[1] = (met->lon[ci[1] + 1] - lon)
01810             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01811         cw[2] = (met->lat[ci[2] + 1] - lat)
01812             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01813     }
01814
01815     /* Interpolate vertically... */
01816     double aux00 =
01817         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01818         + array[ci[1]][ci[2]][ci[0] + 1];
01819     double aux01 =
01820         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01821             array[ci[1]][ci[2] + 1][ci[0] + 1])
01822         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01823     double aux10 =
01824         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01825             array[ci[1] + 1][ci[2]][ci[0] + 1])
01826         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01827     double aux11 =
01828         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01829             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01830         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01831
01832     /* Interpolate horizontally... */
01833     aux00 = cw[2] * (aux00 - aux01) + aux01;
01834     aux11 = cw[2] * (aux10 - aux11) + aux11;
01835     *var = cw[1] * (aux00 - aux11) + aux11;
01836 }
01837

```



```

01838
01839 /*****
01840
01841 void intpol_met_space_2d(
01842     met_t * met,
01843     float array[EX][EY],
01844     double lon,
01845     double lat,
01846     double *var,
01847     int *ci,
01848     double *cw,
01849     int init) {
01850
01851     /* Initialize interpolation... */
01852     if (init) {
01853
01854         /* Check longitude... */
01855         if (met->lon[met->nx - 1] > 180 && lon < 0)
01856             lon += 360;
01857
01858         /* Get interpolation indices... */
01859         ci[1] = locate_reg(met->lon, met->nx, lon);
01860         ci[2] = locate_reg(met->lat, met->ny, lat);
01861
01862         /* Get interpolation weights... */
01863         cw[1] = (met->lon[ci[1] + 1] - lon)
01864             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01865         cw[2] = (met->lat[ci[2] + 1] - lat)
01866             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01867     }
01868
01869     /* Set variables... */
01870     double aux00 = array[ci[1]][ci[2]];
01871     double aux01 = array[ci[1]][ci[2] + 1];
01872     double aux10 = array[ci[1] + 1][ci[2]];
01873     double aux11 = array[ci[1] + 1][ci[2] + 1];
01874
01875     /* Interpolate horizontally... */
01876     if (isfinite(aux00) && isfinite(aux01)
01877         && isfinite(aux10) && isfinite(aux11)) {
01878         aux00 = cw[2] * (aux00 - aux01) + aux01;
01879         aux11 = cw[2] * (aux10 - aux11) + aux11;
01880         *var = cw[1] * (aux00 - aux11) + aux11;
01881     } else {
01882         if (cw[2] < 0.5) {
01883             if (cw[1] < 0.5)
01884                 *var = aux11;
01885             else
01886                 *var = aux01;
01887         } else {
01888             if (cw[1] < 0.5)
01889                 *var = aux10;
01890             else
01891                 *var = aux00;
01892         }
01893     }
01894 }
01895
01896 /*****
01897
01898 void intpol_met_time_3d(
01899     met_t * met0,
01900     float array0[EX][EY][EP],
01901     met_t * met1,
01902     float array1[EX][EY][EP],
01903     double ts,
01904     double p,
01905     double lon,
01906     double lat,
01907     double *var,
01908     int *ci,
01909     double *cw,
01910     int init) {
01911
01912     double var0, var1, wt;
01913
01914     /* Spatial interpolation... */
01915     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01916     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01917
01918     /* Get weighting factor... */
01919     wt = (met1->time - ts) / (met1->time - met0->time);
01920
01921     /* Interpolate... */
01922     *var = wt * (var0 - var1) + var1;
01923 }
01924

```

```

01925 /*****
01926
01927 void intpol_met_time_2d(
01928     met_t * met0,
01929     float array0[EX][EY],
01930     met_t * met1,
01931     float array1[EX][EY],
01932     double ts,
01933     double lon,
01934     double lat,
01935     double *var,
01936     int *ci,
01937     double *cw,
01938     int init) {
01939
01940     double var0, var1, wt;
01941
01942     /* Spatial interpolation... */
01943     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01944     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01945
01946     /* Get weighting factor... */
01947     wt = (met1->time - ts) / (met1->time - met0->time);
01948
01949     /* Interpolate... */
01950     if (isfinite(var0) && isfinite(var1))
01951         *var = wt * (var0 - var1) + var1;
01952     else if (wt < 0.5)
01953         *var = var1;
01954     else
01955         *var = var0;
01956 }
01957
01958 /*****
01959
01960 void jsec2time(
01961     double jsec,
01962     int *year,
01963     int *mon,
01964     int *day,
01965     int *hour,
01966     int *min,
01967     int *sec,
01968     double *remain) {
01969
01970     struct tm t0, *t1;
01971
01972     t0.tm_year = 100;
01973     t0.tm_mon = 0;
01974     t0.tm_mday = 1;
01975     t0.tm_hour = 0;
01976     t0.tm_min = 0;
01977     t0.tm_sec = 0;
01978
01979     time_t jsec0 = (time_t) jsec + timegm(&t0);
01980     t1 = gmtime(&jsec0);
01981
01982     *year = t1->tm_year + 1900;
01983     *mon = t1->tm_mon + 1;
01984     *day = t1->tm_mday;
01985     *hour = t1->tm_hour;
01986     *min = t1->tm_min;
01987     *sec = t1->tm_sec;
01988     *remain = jsec - floor(jsec);
01989 }
01990
01991 /*****
01992
01993 double lapse_rate(
01994     double t,
01995     double h2o) {
01996
01997     /*
01998      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01999      and water vapor volume mixing ratio [1].
02000
02001      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
02002     */
02003
02004     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
02005
02006     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
02007 }
02008
02009 /*****
02010
02011 int locate_irr(

```

```

02012 double *xx,
02013 int n,
02014 double x) {
02015
02016     int ilo = 0;
02017     int ihi = n - 1;
02018     int i = (ihi + ilo) » 1;
02019
02020     if (xx[i] < xx[i + 1])
02021         while (ihi > ilo + 1) {
02022             i = (ihi + ilo) » 1;
02023             if (xx[i] > x)
02024                 ihi = i;
02025             else
02026                 ilo = i;
02027         } else
02028         while (ihi > ilo + 1) {
02029             i = (ihi + ilo) » 1;
02030             if (xx[i] <= x)
02031                 ihi = i;
02032             else
02033                 ilo = i;
02034         }
02035
02036     return ilo;
02037 }
02038
02039 /*****
02040
02041 int locate_reg(
02042     double *xx,
02043     int n,
02044     double x) {
02045
02046     /* Calculate index... */
02047     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
02048
02049     /* Check range... */
02050     if (i < 0)
02051         return 0;
02052     else if (i > n - 2)
02053         return n - 2;
02054     else
02055         return i;
02056 }
02057
02058 /*****
02059
02060 double nat_temperature(
02061     double p,
02062     double h2o,
02063     double hno3) {
02064
02065     /* Check water vapor vmr... */
02066     h2o = GSL_MAX(h2o, 0.1e-6);
02067
02068     /* Calculate T_NAT... */
02069     double p_hno3 = hno3 * p / 1.333224;
02070     double p_h2o = h2o * p / 1.333224;
02071     double a = 0.009179 - 0.00088 * log10(p_h2o);
02072     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
02073     double c = -11397.0 / a;
02074     double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
02075     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
02076     if (x2 > 0)
02077         tnat = x2;
02078
02079     return tnat;
02080 }
02081
02082 /*****
02083
02084 int read_atm(
02085     const char *filename,
02086     ctl_t *ctl,
02087     atm_t *atm) {
02088
02089     FILE *in;
02090
02091     double t0;
02092
02093     int dimid, ncid, varid;
02094
02095     size_t nparts;
02096
02097     /* Set timer... */
02098     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);

```

```

02099
02100  /* Init... */
02101  atm->np = 0;
02102
02103  /* Write info... */
02104  LOG(1, "Read atmospheric data: %s", filename);
02105
02106  /* Read ASCII data... */
02107  if (ctl->atm_type == 0) {
02108
02109      /* Open file... */
02110      if (!(in = fopen(filename, "r"))) {
02111          WARN("File not found!");
02112          return 0;
02113      }
02114
02115      /* Read line... */
02116      char line[LEN];
02117      while (fgets(line, LEN, in)) {
02118
02119          /* Read data... */
02120          char *tok;
02121          TOK(line, tok, "%lg", atm->time[atm->np]);
02122          TOK(NULL, tok, "%lg", atm->p[atm->np]);
02123          TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02124          TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02125          for (int iq = 0; iq < ctl->nq; iq++)
02126              TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02127
02128          /* Convert altitude to pressure... */
02129          atm->p[atm->np] = P(atm->p[atm->np]);
02130
02131          /* Increment data point counter... */
02132          if (++atm->np > NP)
02133              ERRMSG("Too many data points!");
02134      }
02135
02136      /* Close file... */
02137      fclose(in);
02138  }
02139
02140  /* Read binary data... */
02141  else if (ctl->atm_type == 1) {
02142
02143      /* Open file... */
02144      if (!(in = fopen(filename, "r")))
02145          return 0;
02146
02147      /* Read data... */
02148      FREAD(&atm->np, int,
02149          1,
02150          in);
02151      FREAD(atm->time, double,
02152          (size_t) atm->np,
02153          in);
02154      FREAD(atm->p, double,
02155          (size_t) atm->np,
02156          in);
02157      FREAD(atm->lon, double,
02158          (size_t) atm->np,
02159          in);
02160      FREAD(atm->lat, double,
02161          (size_t) atm->np,
02162          in);
02163      for (int iq = 0; iq < ctl->nq; iq++)
02164          FREAD(atm->q[iq], double,
02165              (size_t) atm->np,
02166              in);
02167
02168      /* Close file... */
02169      fclose(in);
02170  }
02171
02172  /* Read netCDF data... */
02173  else if (ctl->atm_type == 2) {
02174
02175      /* Open file... */
02176      if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02177          return 0;
02178
02179      /* Get dimensions... */
02180      NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02181      NC(nc_inq_dimlen(ncid, dimid, &nparts));
02182      atm->np = (int) nparts;
02183      if (atm->np > NP)
02184          ERRMSG("Too many particles!");
02185

```

```

02186  /* Get time... */
02187  NC(nc_inq_varid(ncid, "time", &varid));
02188  NC(nc_get_var_double(ncid, varid, &t0));
02189  for (int ip = 0; ip < atm->np; ip++)
02190      atm->time[ip] = t0;
02191
02192  /* Read geolocations... */
02193  NC(nc_inq_varid(ncid, "PRESS", &varid));
02194  NC(nc_get_var_double(ncid, varid, atm->p));
02195  NC(nc_inq_varid(ncid, "LON", &varid));
02196  NC(nc_get_var_double(ncid, varid, atm->lon));
02197  NC(nc_inq_varid(ncid, "LAT", &varid));
02198  NC(nc_get_var_double(ncid, varid, atm->lat));
02199
02200  /* Read variables... */
02201  if (ctl->qnt_p >= 0)
02202      if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02203          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02204  if (ctl->qnt_t >= 0)
02205      if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02206          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02207  if (ctl->qnt_u >= 0)
02208      if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02209          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02210  if (ctl->qnt_v >= 0)
02211      if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02212          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02213  if (ctl->qnt_w >= 0)
02214      if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02215          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02216  if (ctl->qnt_h2o >= 0)
02217      if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02218          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02219  if (ctl->qnt_o3 >= 0)
02220      if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02221          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02222  if (ctl->qnt_theta >= 0)
02223      if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02224          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02225  if (ctl->qnt_pv >= 0)
02226      if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02227          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02228
02229  /* Check data... */
02230  for (int ip = 0; ip < atm->np; ip++)
02231      if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02232          || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02233          || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02234          || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02235          || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02236      atm->time[ip] = GSL_NAN;
02237      atm->p[ip] = GSL_NAN;
02238      atm->lon[ip] = GSL_NAN;
02239      atm->lat[ip] = GSL_NAN;
02240      for (int iq = 0; iq < ctl->nq; iq++)
02241          atm->q[iq][ip] = GSL_NAN;
02242      } else {
02243          if (ctl->qnt_h2o >= 0)
02244              atm->q[ctl->qnt_h2o][ip] *= 1.608;
02245          if (ctl->qnt_pv >= 0)
02246              atm->q[ctl->qnt_pv][ip] *= 1e6;
02247          if (atm->lon[ip] > 180)
02248              atm->lon[ip] -= 360;
02249      }
02250
02251  /* Close file... */
02252  NC(nc_close(ncid));
02253  }
02254
02255  /* Error... */
02256  else
02257      ERRMSG("Atmospheric data type not supported!");
02258
02259  /* Check number of points... */
02260  if (atm->np < 1)
02261      ERRMSG("Can not read any data!");
02262
02263  /* Write info... */
02264  double mini, maxi;
02265  LOG(2, "Number of particles: %d", atm->np);
02266  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
02267  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
02268  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
02269  LOG(2, "Altitude range: %g ... %g km", Z(mini), Z(maxi));
02270  LOG(2, "Pressure range: %g ... %g hPa", mini, maxi);
02271  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
02272  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);

```

```

02273     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
02274     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
02275     for (int iq = 0; iq < ctl->nq; iq++) {
02276         char msg[LEN];
02277         sprintf(msg, "Quantity %s range: %s ... %s %s",
02278                 ctl->qnt_name[iq], ctl->qnt_format[iq],
02279                 ctl->qnt_format[iq], ctl->qnt_unit[iq]);
02280         gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
02281         LOG(2, msg, mini, maxi);
02282     }
02283
02284     /* Return success... */
02285     return 1;
02286 }
02287
02288 /*****
02289
02290 void read_ctl(
02291     const char *filename,
02292     int argc,
02293     char *argv[],
02294     ctl_t * ctl) {
02295
02296     /* Set timer... */
02297     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02298
02299     /* Write info... */
02300     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02301           "(executable: %s | version: %s | compiled: %s, %s)\n",
02302         argv[0], VERSION, __DATE__, __TIME__);
02303
02304     /* Initialize quantity indices... */
02305     ctl->qnt_ens = -1;
02306     ctl->qnt_stat = -1;
02307     ctl->qnt_m = -1;
02308     ctl->qnt_vmr = -1;
02309     ctl->qnt_r = -1;
02310     ctl->qnt_rho = -1;
02311     ctl->qnt_ps = -1;
02312     ctl->qnt_ts = -1;
02313     ctl->qnt_zs = -1;
02314     ctl->qnt_us = -1;
02315     ctl->qnt_vs = -1;
02316     ctl->qnt_pbl = -1;
02317     ctl->qnt_pt = -1;
02318     ctl->qnt_tt = -1;
02319     ctl->qnt_zt = -1;
02320     ctl->qnt_h2ot = -1;
02321     ctl->qnt_z = -1;
02322     ctl->qnt_p = -1;
02323     ctl->qnt_t = -1;
02324     ctl->qnt_u = -1;
02325     ctl->qnt_v = -1;
02326     ctl->qnt_w = -1;
02327     ctl->qnt_h2o = -1;
02328     ctl->qnt_o3 = -1;
02329     ctl->qnt_lwc = -1;
02330     ctl->qnt_iwc = -1;
02331     ctl->qnt_pct = -1;
02332     ctl->qnt_pcb = -1;
02333     ctl->qnt_cl = -1;
02334     ctl->qnt_plcl = -1;
02335     ctl->qnt_plfc = -1;
02336     ctl->qnt_pel = -1;
02337     ctl->qnt_cape = -1;
02338     ctl->qnt_cin = -1;
02339     ctl->qnt_hno3 = -1;
02340     ctl->qnt_oh = -1;
02341     ctl->qnt_psat = -1;
02342     ctl->qnt_psize = -1;
02343     ctl->qnt_pw = -1;
02344     ctl->qnt_sh = -1;
02345     ctl->qnt_rh = -1;
02346     ctl->qnt_rhice = -1;
02347     ctl->qnt_theta = -1;
02348     ctl->qnt_zeta = -1;
02349     ctl->qnt_tvirt = -1;
02350     ctl->qnt_lapse = -1;
02351     ctl->qnt_vh = -1;
02352     ctl->qnt_vz = -1;
02353     ctl->qnt_pv = -1;
02354     ctl->qnt_tdew = -1;
02355     ctl->qnt_tice = -1;
02356     ctl->qnt_tsts = -1;
02357     ctl->qnt_tnat = -1;
02358
02359     /* Read quantities... */

```

```

02360     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02361     if (ctl->nq > NQ)
02362         ERRMSG("Too many quantities!");
02363     for (int iq = 0; iq < ctl->nq; iq++) {
02364
02365         /* Read quantity name and format... */
02366         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02367         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02368             ctl->qnt_format[iq]);
02369
02370         /* Try to identify quantity... */
02371         SET_QNT(qnt_ens, "ens", "-")
02372         SET_QNT(qnt_stat, "stat", "-")
02373         SET_QNT(qnt_m, "m", "kg")
02374         SET_QNT(qnt_vmr, "vmr", "ppv")
02375         SET_QNT(qnt_r, "r", "microns")
02376         SET_QNT(qnt_rho, "rho", "kg/m^3")
02377         SET_QNT(qnt_ps, "ps", "hPa")
02378         SET_QNT(qnt_ts, "ts", "K")
02379         SET_QNT(qnt_zs, "zs", "km")
02380         SET_QNT(qnt_us, "us", "m/s")
02381         SET_QNT(qnt_vs, "vs", "m/s")
02382         SET_QNT(qnt_pbl, "pbl", "hPa")
02383         SET_QNT(qnt_pt, "pt", "hPa")
02384         SET_QNT(qnt_tt, "tt", "K")
02385         SET_QNT(qnt_zt, "zt", "km")
02386         SET_QNT(qnt_h2ot, "h2ot", "ppv")
02387         SET_QNT(qnt_z, "z", "km")
02388         SET_QNT(qnt_p, "p", "hPa")
02389         SET_QNT(qnt_t, "t", "K")
02390         SET_QNT(qnt_u, "u", "m/s")
02391         SET_QNT(qnt_v, "v", "m/s")
02392         SET_QNT(qnt_w, "w", "hPa/s")
02393         SET_QNT(qnt_h2o, "h2o", "ppv")
02394         SET_QNT(qnt_o3, "o3", "ppv")
02395         SET_QNT(qnt_lwc, "lwc", "kg/kg")
02396         SET_QNT(qnt_iwc, "iwc", "kg/kg")
02397         SET_QNT(qnt_pct, "pct", "hPa")
02398         SET_QNT(qnt_pcb, "pcb", "hPa")
02399         SET_QNT(qnt_cl, "cl", "kg/m^2")
02400         SET_QNT(qnt_plcl, "plcl", "hPa")
02401         SET_QNT(qnt_plfc, "plfc", "hPa")
02402         SET_QNT(qnt_pel, "pel", "hPa")
02403         SET_QNT(qnt_cape, "cape", "J/kg")
02404         SET_QNT(qnt_cin, "cin", "J/kg")
02405         SET_QNT(qnt_hno3, "hno3", "ppv")
02406         SET_QNT(qnt_oh, "oh", "molec/cm^3")
02407         SET_QNT(qnt_psat, "psat", "hPa")
02408         SET_QNT(qnt_psice, "psice", "hPa")
02409         SET_QNT(qnt_pw, "pw", "hPa")
02410         SET_QNT(qnt_sh, "sh", "kg/kg")
02411         SET_QNT(qnt_rh, "rh", "%")
02412         SET_QNT(qnt_rhice, "rhice", "%")
02413         SET_QNT(qnt_theta, "theta", "K")
02414         SET_QNT(qnt_zeta, "zeta", "K")
02415         SET_QNT(qnt_tvirt, "tvirt", "K")
02416         SET_QNT(qnt_lapse, "lapse", "K/km")
02417         SET_QNT(qnt_vh, "vh", "m/s")
02418         SET_QNT(qnt_vz, "vz", "m/s")
02419         SET_QNT(qnt_pv, "pv", "PVU")
02420         SET_QNT(qnt_tdew, "tdew", "K")
02421         SET_QNT(qnt_tice, "tice", "K")
02422         SET_QNT(qnt_tsts, "tsts", "K")
02423         SET_QNT(qnt_tnat, "tnat", "K")
02424         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02425     }
02426
02427     /* netCDF I/O parameters... */
02428     ctl->chunkszhint =
02429         (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02430             NULL);
02431     ctl->read_mode =
02432         (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02433
02434     /* Time steps of simulation... */
02435     ctl->direction =
02436         (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02437     if (ctl->direction != -1 && ctl->direction != 1)
02438         ERRMSG("Set DIRECTION to -1 or 1!");
02439     ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02440     ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02441
02442     /* Meteorological data... */
02443     scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02444     ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02445     ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02446     ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);

```

```

02447 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02448 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02449     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02450 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02451 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02452 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02453 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02454     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02455 ctl->met_detrend =
02456     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02457 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02458 if (ctl->met_np > EP)
02459     ERRMSG("Too many levels!");
02460 for (int ip = 0; ip < ctl->met_np; ip++)
02461     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02462 ctl->met_geopot_sx
02463     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02464 ctl->met_geopot_sy
02465     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02466 ctl->met_tropo =
02467     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02468 if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02469     ERRMSG("Set MET_TROPO = 0 ... 5!");
02470 ctl->met_tropo_lapse =
02471     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02472 ctl->met_tropo_nlev =
02473     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02474 ctl->met_tropo_lapse_sep =
02475     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02476 ctl->met_tropo_nlev_sep =
02477     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02478     NULL);
02479 ctl->met_tropo_pv =
02480     scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02481 ctl->met_tropo_theta =
02482     scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02483 ctl->met_tropo_spline =
02484     (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02485 ctl->met_cloud =
02486     (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02487 if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02488     ERRMSG("Set MET_CLOUD = 0 ... 3!");
02489 ctl->met_dt_out =
02490     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02491 ctl->met_cache =
02492     (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02493
02494 /* Isosurface parameters... */
02495 ctl->isosurf =
02496     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02497 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02498
02499 /* Advection parameters... */
02500 ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "0", NULL);
02501 ctl->reflect =
02502     (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02503
02504 /* Diffusion parameters... */
02505 ctl->turb_dx_trop =
02506     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02507 ctl->turb_dx_strat =
02508     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02509 ctl->turb_dz_trop =
02510     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02511 ctl->turb_dz_strat =
02512     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02513 ctl->turb_mesox =
02514     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02515 ctl->turb_mesoz =
02516     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02517
02518 /* Convection... */
02519 ctl->conv_cape
02520     = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02521 ctl->conv_cin
02522     = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02523 ctl->conv_wmax
02524     = scan_ctl(filename, argc, argv, "CONV_WMAX", -1, "-999", NULL);
02525 ctl->conv_wcape
02526     = (int) scan_ctl(filename, argc, argv, "CONV_WCAPE", -1, "0", NULL);
02527 ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02528 ctl->conv_mix_bot
02529     = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02530 ctl->conv_mix_top
02531     = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02532
02533 /* Boundary conditions... */

```



```
02534     ctl->bound_mass =
02535         scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02536     ctl->bound_vmr =
02537         scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02538     ctl->bound_lat0 =
02539         scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02540     ctl->bound_lat1 =
02541         scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02542     ctl->bound_p0 =
02543         scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02544     ctl->bound_p1 =
02545         scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02546     ctl->bound_dps =
02547         scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02548
02549     /* Species parameters... */
02550     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02551     if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02552         ctl->molmass = 120.907;
02553         ctl->wet_depo[2] = ctl->wet_depo[6] = 3e-5;
02554         ctl->wet_depo[3] = ctl->wet_depo[7] = 3500.0;
02555     } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02556         ctl->molmass = 137.359;
02557         ctl->wet_depo[2] = ctl->wet_depo[6] = 1.1e-4;
02558         ctl->wet_depo[3] = ctl->wet_depo[7] = 3300.0;
02559     } else if (strcasecmp(ctl->species, "CH4") == 0) {
02560         ctl->molmass = 16.043;
02561         ctl->oh_chem_reaction = 2;
02562         ctl->oh_chem[0] = 2.45e-12;
02563         ctl->oh_chem[1] = 1775;
02564         ctl->wet_depo[2] = ctl->wet_depo[6] = 1.4e-5;
02565         ctl->wet_depo[3] = ctl->wet_depo[7] = 1600.0;
02566     } else if (strcasecmp(ctl->species, "CO") == 0) {
02567         ctl->molmass = 28.01;
02568         ctl->oh_chem_reaction = 3;
02569         ctl->oh_chem[0] = 6.9e-33;
02570         ctl->oh_chem[1] = 2.1;
02571         ctl->oh_chem[2] = 1.1e-12;
02572         ctl->oh_chem[3] = -1.3;
02573         ctl->wet_depo[2] = ctl->wet_depo[6] = 9.7e-6;
02574         ctl->wet_depo[3] = ctl->wet_depo[7] = 1300.0;
02575     } else if (strcasecmp(ctl->species, "CO2") == 0) {
02576         ctl->molmass = 44.009;
02577         ctl->wet_depo[2] = ctl->wet_depo[6] = 3.3e-4;
02578         ctl->wet_depo[3] = ctl->wet_depo[7] = 2400.0;
02579     } else if (strcasecmp(ctl->species, "N2O") == 0) {
02580         ctl->molmass = 44.013;
02581         ctl->wet_depo[2] = ctl->wet_depo[6] = 2.4e-4;
02582         ctl->wet_depo[3] = ctl->wet_depo[7] = 2600.;
02583     } else if (strcasecmp(ctl->species, "NH3") == 0) {
02584         ctl->molmass = 17.031;
02585         ctl->oh_chem_reaction = 2;
02586         ctl->oh_chem[0] = 1.7e-12;
02587         ctl->oh_chem[1] = 710;
02588         ctl->wet_depo[2] = ctl->wet_depo[6] = 5.9e-1;
02589         ctl->wet_depo[3] = ctl->wet_depo[7] = 4200.0;
02590     } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02591         ctl->molmass = 63.012;
02592         ctl->wet_depo[2] = ctl->wet_depo[6] = 2.1e3;
02593         ctl->wet_depo[3] = ctl->wet_depo[7] = 8700.0;
02594     } else if (strcasecmp(ctl->species, "NO") == 0) {
02595         ctl->molmass = 30.006;
02596         ctl->oh_chem_reaction = 3;
02597         ctl->oh_chem[0] = 7.1e-31;
02598         ctl->oh_chem[1] = 2.6;
02599         ctl->oh_chem[2] = 3.6e-11;
02600         ctl->oh_chem[3] = 0.1;
02601         ctl->wet_depo[2] = ctl->wet_depo[6] = 1.9e-5;
02602         ctl->wet_depo[3] = ctl->wet_depo[7] = 1600.0;
02603     } else if (strcasecmp(ctl->species, "NO2") == 0) {
02604         ctl->molmass = 46.005;
02605         ctl->oh_chem_reaction = 3;
02606         ctl->oh_chem[0] = 1.8e-30;
02607         ctl->oh_chem[1] = 3.0;
02608         ctl->oh_chem[2] = 2.8e-11;
02609         ctl->oh_chem[3] = 0.0;
02610         ctl->wet_depo[2] = ctl->wet_depo[6] = 1.2e-4;
02611         ctl->wet_depo[3] = ctl->wet_depo[7] = 2400.0;
02612     } else if (strcasecmp(ctl->species, "O3") == 0) {
02613         ctl->molmass = 47.997;
02614         ctl->oh_chem_reaction = 2;
02615         ctl->oh_chem[0] = 1.7e-12;
02616         ctl->oh_chem[1] = 940;
02617         ctl->wet_depo[2] = ctl->wet_depo[6] = 1e-4;
02618         ctl->wet_depo[3] = ctl->wet_depo[7] = 2800.0;
02619     } else if (strcasecmp(ctl->species, "SF6") == 0) {
02620         ctl->molmass = 146.048;
```

```

02621     ctl->wet_depo[2] = ctl->wet_depo[6] = 2.4e-6;
02622     ctl->wet_depo[3] = ctl->wet_depo[7] = 3100.0;
02623 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02624     ctl->molmass = 64.066;
02625     ctl->oh_chem_reaction = 3;
02626     ctl->oh_chem[0] = 2.9e-31;
02627     ctl->oh_chem[1] = 4.1;
02628     ctl->oh_chem[2] = 1.7e-12;
02629     ctl->oh_chem[3] = -0.2;
02630     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.3e-2;
02631     ctl->wet_depo[3] = ctl->wet_depo[7] = 2900.0;
02632 } else {
02633     ctl->molmass =
02634         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02635     ctl->oh_chem_reaction =
02636         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02637     for (int ip = 0; ip < 4; ip++)
02638         ctl->oh_chem[ip] =
02639             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02640     for (int ip = 0; ip < 1; ip++)
02641         ctl->dry_depo[ip] =
02642             scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02643     for (int ip = 0; ip < 8; ip++)
02644         ctl->wet_depo[ip] =
02645             scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02646 }
02647 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02648 ctl->tdec_strat =
02649     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02650
02651 /* PSC analysis... */
02652 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02653 ctl->psc_hno3 =
02654     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02655
02656 /* Output of atmospheric data... */
02657 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02658 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02659 ctl->atm_dt_out =
02660     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02661 ctl->atm_filter =
02662     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02663 ctl->atm_stride =
02664     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02665 ctl->atm_type =
02666     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02667
02668 /* Output of CSI data... */
02669 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02670 ctl->csi_dt_out =
02671     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02672 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02673 ctl->csi_obsmin =
02674     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02675 ctl->csi_modmin =
02676     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02677 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02678 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02679 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02680 ctl->csi_lon0 =
02681     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02682 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02683 ctl->csi_nx =
02684     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02685 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02686 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02687 ctl->csi_ny =
02688     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02689
02690 /* Output of ensemble data... */
02691 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02692
02693 /* Output of grid data... */
02694 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02695         ctl->grid_basename);
02696 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02697 ctl->grid_dt_out =
02698     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02699 ctl->grid_sparse =
02700     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02701 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02702 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02703 ctl->grid_nz =
02704     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02705 ctl->grid_lon0 =
02706     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02707 ctl->grid_lon1 =

```

```

02708     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02709     ctl->grid_nx =
02710         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02711     ctl->grid_lat0 =
02712         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02713     ctl->grid_lat1 =
02714         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02715     ctl->grid_ny =
02716         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02717
02718     /* Output of profile data... */
02719     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02720         ctl->prof_basename);
02721     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02722     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02723     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02724     ctl->prof_nz =
02725         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02726     ctl->prof_lon0 =
02727         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02728     ctl->prof_lon1 =
02729         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02730     ctl->prof_nx =
02731         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02732     ctl->prof_lat0 =
02733         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02734     ctl->prof_lat1 =
02735         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02736     ctl->prof_ny =
02737         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02738
02739     /* Output of sample data... */
02740     scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02741         ctl->sample_basename);
02742     scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02743         ctl->sample_obsfile);
02744     ctl->sample_dx =
02745         scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02746     ctl->sample_dz =
02747         scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02748
02749     /* Output of station data... */
02750     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02751         ctl->stat_basename);
02752     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02753     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02754     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02755     ctl->stat_t0 =
02756         scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02757     ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02758 }
02759
02760 /*****
02761
02762 int read_met(
02763     ctl_t * ctl,
02764     char *filename,
02765     met_t * met) {
02766
02767     int ncid;
02768
02769     /* Write info... */
02770     LOG(1, "Read meteorological data: %s", filename);
02771
02772     /* Open netCDF file... */
02773     if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02774         NC_NOERR) {
02775         WARN("File not found!");
02776         return 0;
02777     }
02778
02779     /* Read coordinates of meteorological data... */
02780     read_met_grid(filename, ncid, ctl, met);
02781
02782     /* Read meteo data on vertical levels... */
02783     read_met_levels(ncid, ctl, met);
02784
02785     /* Extrapolate data for lower boundary... */
02786     read_met_extrapolate(met);
02787
02788     /* Read surface data... */
02789     read_met_surface(ncid, met);
02790
02791     /* Create periodic boundary conditions... */
02792     read_met_periodic(met);
02793
02794     /* Downsampling... */

```

```

02795     read_met_sample(ctl, met);
02796
02797     /* Calculate geopotential heights... */
02798     read_met_geopot(ctl, met);
02799
02800     /* Calculate potential vorticity... */
02801     read_met_pv(met);
02802
02803     /* Calculate boundary layer data... */
02804     read_met_pbl(met);
02805
02806     /* Calculate tropopause data... */
02807     read_met_tropo(ctl, met);
02808
02809     /* Calculate cloud properties... */
02810     read_met_cloud(met);
02811
02812     /* Calculate convective available potential energy... */
02813     read_metCAPE(met);
02814
02815     /* Detrending... */
02816     read_met_detrend(ctl, met);
02817
02818     /* Close file... */
02819     NC(nc_close(ncid));
02820
02821     /* Return success... */
02822     return 1;
02823 }
02824
02825 /*****
02826
02827 void read_metCAPE(
02828     met_t * met) {
02829
02830     /* Set timer... */
02831     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
02832     LOG(2, "Calculate CAPE...");
02833
02834     /* Vertical spacing (about 100 m)... */
02835     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
02836
02837     /* Loop over columns... */
02838     #pragma omp parallel for default(shared) collapse(2)
02839     for (int ix = 0; ix < met->nx; ix++)
02840         for (int iy = 0; iy < met->ny; iy++) {
02841
02842             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
02843             int n = 0;
02844             double h2o = 0, t, theta = 0;
02845             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
02846             double ptop = pbot - 50.;
02847             for (int ip = 0; ip < met->np; ip++) {
02848                 if (met->p[ip] <= pbot) {
02849                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
02850                     h2o += met->h2o[ix][iy][ip];
02851                     n++;
02852                 }
02853                 if (met->p[ip] < ptop && n > 0)
02854                     break;
02855             }
02856             theta /= n;
02857             h2o /= n;
02858
02859             /* Cannot compute anything if water vapor is missing... */
02860             met->plcl[ix][iy] = GSL_NAN;
02861             met->plfc[ix][iy] = GSL_NAN;
02862             met->pel[ix][iy] = GSL_NAN;
02863             met->CAPE[ix][iy] = GSL_NAN;
02864             met->cin[ix][iy] = GSL_NAN;
02865             if (h2o <= 0)
02866                 continue;
02867
02868             /* Find lifted condensation level (LCL)... */
02869             ptop = P(20.);
02870             pbot = met->ps[ix][iy];
02871             do {
02872                 met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
02873                 t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
02874                 if (RH(met->plcl[ix][iy], t, h2o) > 100.)
02875                     ptop = met->plcl[ix][iy];
02876                 else
02877                     pbot = met->plcl[ix][iy];
02878             } while (pbot - ptop > 0.1);
02879
02880             /* Calculate CIN up to LCL... */
02881             INTPOL_INIT;

```

```

02882 double dcape, dcape_old, dz, psat, h2o_env, t_env;
02883 double p = met->ps[ix][iy];
02884 met->cape[ix][iy] = met->cin[ix][iy] = 0;
02885 do {
02886     dz = dz0 * TVIRT(t, h2o);
02887     p /= pfac;
02888     t = theta / pow(1000. / p, 0.286);
02889     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02890         &t_env, ci, cw, 1);
02891     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02892         &h2o_env, ci, cw, 0);
02893     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02894         TVIRT(t_env, h2o_env) * dz;
02895     if (dcape < 0)
02896         met->cin[ix][iy] += fabsf((float) dcape);
02897 } while (p > met->plcl[ix][iy]);

02898
02899 /* Calculate level of free convection (LFC), equilibrium level (EL),
02900    and convective available potential energy (CAPE)... */
02901 dcape = 0;
02902 p = met->plcl[ix][iy];
02903 t = theta / pow(1000. / p, 0.286);
02904 ptop = 0.75 * clim_tropo(met->time, met->lat[iy]);
02905 do {
02906     dz = dz0 * TVIRT(t, h2o);
02907     p /= pfac;
02908     t -= lapse_rate(t, h2o) * dz;
02909     psat = PSAT(t);
02910     h2o = psat / (p - (1. - EPS) * psat);
02911     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02912         &t_env, ci, cw, 1);
02913     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02914         &h2o_env, ci, cw, 0);
02915     dcape_old = dcape;
02916     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02917         TVIRT(t_env, h2o_env) * dz;
02918     if (dcape > 0) {
02919         met->cape[ix][iy] += (float) dcape;
02920         if (!isfinite(met->plfc[ix][iy]))
02921             met->plfc[ix][iy] = (float) p;
02922     } else if (dcape_old > 0)
02923         met->pel[ix][iy] = (float) p;
02924     if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
02925         met->cin[ix][iy] += fabsf((float) dcape);
02926 } while (p > ptop);
02927
02928 /* Check results... */
02929 if (!isfinite(met->plfc[ix][iy]))
02930     met->cin[ix][iy] = GSL_NAN;
02931 }
02932 }
02933
02934 /*****
02935
02936 void read_met_cloud(
02937     met_t * met) {
02938
02939     /* Set timer... */
02940     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
02941     LOG(2, "Calculate cloud data...");
02942
02943     /* Loop over columns... */
02944     #pragma omp parallel for default(shared) collapse(2)
02945     for (int ix = 0; ix < met->nx; ix++)
02946         for (int iy = 0; iy < met->ny; iy++) {
02947
02948             /* Init... */
02949             met->pct[ix][iy] = GSL_NAN;
02950             met->pcb[ix][iy] = GSL_NAN;
02951             met->cl[ix][iy] = 0;
02952
02953             /* Loop over pressure levels... */
02954             for (int ip = 0; ip < met->np - 1; ip++) {
02955
02956                 /* Check pressure... */
02957                 if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02958                     continue;
02959
02960                 /* Check ice water and liquid water content... */
02961                 if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0) {
02962
02963                     /* Get cloud top pressure ... */
02964                     met->pct[ix][iy]
02965                         = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
02966
02967                     /* Get cloud bottom pressure ... */
02968                     if (!isfinite(met->pcb[ix][iy]))

```

```

02969         met->pcb[ix][iy]
02970         = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
02971     }
02972
02973     /* Get cloud water... */
02974     met->cl[ix][iy] += (float)
02975     (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02976     + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02977     * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02978 }
02979 }
02980 }
02981
02982 /*****
02983
02984 void read_met_detrend(
02985     ctl_t * ctl,
02986     met_t * met) {
02987
02988     met_t *help;
02989
02990     /* Check parameters... */
02991     if (ctl->met_detrend <= 0)
02992         return;
02993
02994     /* Set timer... */
02995     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
02996     LOG(2, "Detrend meteo data...");
02997
02998     /* Allocate... */
02999     ALLOC(help, met_t, 1);
03000
03001     /* Calculate standard deviation... */
03002     double sigma = ctl->met_detrend / 2.355;
03003     double tssq = 2. * SQR(sigma);
03004
03005     /* Calculate box size in latitude... */
03006     int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03007     sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03008
03009     /* Calculate background... */
03010     #pragma omp parallel for default(shared) collapse(2)
03011     for (int ix = 0; ix < met->nx; ix++) {
03012         for (int iy = 0; iy < met->ny; iy++) {
03013
03014             /* Calculate Cartesian coordinates... */
03015             double x0[3];
03016             geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03017
03018             /* Calculate box size in longitude... */
03019             int sx =
03020             (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03021             fabs(met->lon[1] - met->lon[0]));
03022             sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03023
03024             /* Init... */
03025             float wsum = 0;
03026             for (int ip = 0; ip < met->np; ip++) {
03027                 help->t[ix][iy][ip] = 0;
03028                 help->u[ix][iy][ip] = 0;
03029                 help->v[ix][iy][ip] = 0;
03030                 help->w[ix][iy][ip] = 0;
03031             }
03032
03033             /* Loop over neighboring grid points... */
03034             for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03035                 int ix3 = ix2;
03036                 if (ix3 < 0)
03037                     ix3 += met->nx;
03038                 else if (ix3 >= met->nx)
03039                     ix3 -= met->nx;
03040                 for (int iy2 = GSL_MAX(iy - sy, 0);
03041                     iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03042
03043                     /* Calculate Cartesian coordinates... */
03044                     double x1[3];
03045                     geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03046
03047                     /* Calculate weighting factor... */
03048                     float w = (float) exp(-DIST2(x0, x1) / tssq);
03049
03050                     /* Add data... */
03051                     wsum += w;
03052                     for (int ip = 0; ip < met->np; ip++) {
03053                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03054                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03055                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];

```

```

03056         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03057     }
03058 }
03059 }
03060
03061 /* Normalize... */
03062 for (int ip = 0; ip < met->np; ip++) {
03063     help->t[ix][iy][ip] /= wsum;
03064     help->u[ix][iy][ip] /= wsum;
03065     help->v[ix][iy][ip] /= wsum;
03066     help->w[ix][iy][ip] /= wsum;
03067 }
03068 }
03069 }
03070
03071 /* Subtract background... */
03072 #pragma omp parallel for default(shared) collapse(3)
03073 for (int ix = 0; ix < met->nx; ix++)
03074     for (int iy = 0; iy < met->ny; iy++)
03075         for (int ip = 0; ip < met->np; ip++) {
03076             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03077             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03078             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03079             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03080         }
03081
03082 /* Free... */
03083 free(help);
03084 }
03085
03086 /*****
03087 void read_met_extrapolate(
03088     met_t * met) {
03089
03090     /* Set timer... */
03091     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03092     LOG(2, "Extrapolate meteo data...");
03093
03094     /* Loop over columns... */
03095     #pragma omp parallel for default(shared) collapse(2)
03096     for (int ix = 0; ix < met->nx; ix++)
03097         for (int iy = 0; iy < met->ny; iy++) {
03098
03099             /* Find lowest valid data point... */
03100             int ip0;
03101             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03102                 if (!isfinite(met->t[ix][iy][ip0])
03103                     || !isfinite(met->u[ix][iy][ip0])
03104                     || !isfinite(met->v[ix][iy][ip0])
03105                     || !isfinite(met->w[ix][iy][ip0]))
03106                     break;
03107
03108             /* Extrapolate... */
03109             for (int ip = ip0; ip >= 0; ip--) {
03110                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03111                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03112                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03113                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03114                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03115                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03116                 met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03117                 met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03118             }
03119         }
03120     }
03121 }
03122
03123 /*****
03124 void read_met_geopot(
03125     ctl_t * ctl,
03126     met_t * met) {
03127
03128     static float help[EP][EX][EY];
03129
03130     double logp[EP];
03131
03132     int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03133
03134     /* Set timer... */
03135     SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03136     LOG(2, "Calculate geopotential heights...");
03137
03138     /* Calculate log pressure... */
03139     #pragma omp parallel for default(shared)
03140     for (int ip = 0; ip < met->np; ip++)
03141         logp[ip] = log(met->p[ip]);

```

```

03143
03144  /* Apply hydrostatic equation to calculate geopotential heights... */
03145 #pragma omp parallel for default(shared) collapse(2)
03146 for (int ix = 0; ix < met->nx; ix++)
03147     for (int iy = 0; iy < met->ny; iy++) {
03148
03149         /* Get surface height and pressure... */
03150         double zs = met->zs[ix][iy];
03151         double lnps = log(met->ps[ix][iy]);
03152
03153         /* Get temperature and water vapor vmr at the surface... */
03154         int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03155         double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03156             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03157         double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03158             met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03159
03160         /* Upper part of profile... */
03161         met->z[ix][iy][ip0 + 1]
03162             = (float) (zs +
03163                 ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03164                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03165         for (int ip = ip0 + 2; ip < met->np; ip++)
03166             met->z[ix][iy][ip]
03167                 = (float) (met->z[ix][iy][ip - 1] +
03168                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03169                         met->h2o[ix][iy][ip - 1], logp[ip],
03170                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03171
03172         /* Lower part of profile... */
03173         met->z[ix][iy][ip0]
03174             = (float) (zs +
03175                 ZDIFF(lnps, ts, h2os, logp[ip0],
03176                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03177         for (int ip = ip0 - 1; ip >= 0; ip--)
03178             met->z[ix][iy][ip]
03179                 = (float) (met->z[ix][iy][ip + 1] +
03180                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03181                         met->h2o[ix][iy][ip + 1], logp[ip],
03182                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03183     }
03184
03185     /* Check control parameters... */
03186     if (dx == 0 || dy == 0)
03187         return;
03188
03189     /* Default smoothing parameters... */
03190     if (dx < 0 || dy < 0) {
03191         if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03192             dx = 3;
03193             dy = 2;
03194         } else {
03195             dx = 6;
03196             dy = 4;
03197         }
03198     }
03199
03200     /* Calculate weights for smoothing... */
03201     float ws[dx + 1][dy + 1];
03202 #pragma omp parallel for default(shared) collapse(2)
03203     for (int ix = 0; ix <= dx; ix++)
03204         for (int iy = 0; iy <= dy; iy++)
03205             ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03206                 * (1.0f - (float) iy / (float) dy);
03207
03208     /* Copy data... */
03209 #pragma omp parallel for default(shared) collapse(3)
03210     for (int ix = 0; ix < met->nx; ix++)
03211         for (int iy = 0; iy < met->ny; iy++)
03212             for (int ip = 0; ip < met->np; ip++)
03213                 help[ip][ix][iy] = met->z[ix][iy][ip];
03214
03215     /* Horizontal smoothing... */
03216 #pragma omp parallel for default(shared) collapse(3)
03217     for (int ip = 0; ip < met->np; ip++)
03218         for (int ix = 0; ix < met->nx; ix++)
03219             for (int iy = 0; iy < met->ny; iy++) {
03220                 float res = 0, wsum = 0;
03221                 int iy0 = GSL_MAX(iy - dy + 1, 0);
03222                 int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03223                 for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03224                     int ix3 = ix2;
03225                     if (ix3 < 0)
03226                         ix3 += met->nx;
03227                     else if (ix3 >= met->nx)
03228                         ix3 -= met->nx;
03229                     for (int iy2 = iy0; iy2 <= iy1; ++iy2)

```



```

03230         if (isfinite(help[ip][ix3][iy2])) {
03231             float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03232             res += w * help[ip][ix3][iy2];
03233             wsum += w;
03234         }
03235     }
03236     if (wsum > 0)
03237         met->z[ix][iy][ip] = res / wsum;
03238     else
03239         met->z[ix][iy][ip] = GSL_NAN;
03240 }
03241 }
03242
03243 /*****
03244
03245 void read_met_grid(
03246     char *filename,
03247     int ncid,
03248     ctl_t *ctl,
03249     met_t *met) {
03250
03251     char levname[LEN], tstr[10];
03252
03253     double rtime, r2;
03254
03255     int dimid, varid, year2, mon2, day2, hour2, min2, sec2;
03256
03257     size_t np, nx, ny;
03258
03259     /* Set timer... */
03260     SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03261     LOG(2, "Read meteo grid information...");
03262
03263     /* Get time from filename... */
03264     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
03265     int year = atoi(tstr);
03266     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
03267     int mon = atoi(tstr);
03268     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
03269     int day = atoi(tstr);
03270     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03271     int hour = atoi(tstr);
03272     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03273
03274     /* Check time... */
03275     if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03276         || day < 1 || day > 31 || hour < 0 || hour > 23)
03277         ERRMSG("Cannot read time from filename!");
03278     jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03279     LOG(2, "Time from filename: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03280         met->time, year2, mon2, day2, hour2, min2);
03281
03282     /* Check time information... */
03283     if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03284         NC(nc_get_var_double(ncid, varid, &rtime));
03285         if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
03286             WARN("Time information in meteo file does not match filename!");
03287     } else
03288         WARN("Time information in meteo file is missing!");
03289
03290     /* Get grid dimensions... */
03291     NC(nc_inq_dimid(ncid, "lon", &dimid));
03292     NC(nc_inq_dimlen(ncid, dimid, &nx));
03293     LOG(2, "Number of longitudes: %zu", nx);
03294     if (nx < 2 || nx > EX)
03295         ERRMSG("Number of longitudes out of range!");
03296
03297     NC(nc_inq_dimid(ncid, "lat", &dimid));
03298     NC(nc_inq_dimlen(ncid, dimid, &ny));
03299     LOG(2, "Number of latitudes: %zu", ny);
03300     if (ny < 2 || ny > EY)
03301         ERRMSG("Number of latitudes out of range!");
03302
03303     sprintf(levname, "lev");
03304     NC(nc_inq_dimid(ncid, levname, &dimid));
03305     NC(nc_inq_dimlen(ncid, dimid, &np));
03306     if (np == 1) {
03307         sprintf(levname, "lev_2");
03308         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03309             sprintf(levname, "plev");
03310             nc_inq_dimid(ncid, levname, &dimid);
03311         }
03312         NC(nc_inq_dimlen(ncid, dimid, &np));
03313     }
03314     LOG(2, "Number of levels: %zu", np);
03315     if (np < 2 || np > EP)
03316         ERRMSG("Number of levels out of range!");

```

```

03317
03318 /* Store dimensions... */
03319 met->np = (int) np;
03320 met->nx = (int) nx;
03321 met->ny = (int) ny;
03322
03323 /* Read longitudes and latitudes... */
03324 NC(nc_inq_varid(ncid, "lon", &varid));
03325 NC(nc_get_var_double(ncid, varid, met->lon));
03326 LOG(2, "Longitudes: %g, %g ... %g deg",
03327      met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03328 NC(nc_inq_varid(ncid, "lat", &varid));
03329 NC(nc_get_var_double(ncid, varid, met->lat));
03330 LOG(2, "Latitudes: %g, %g ... %g deg",
03331      met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03332
03333 /* Read pressure levels... */
03334 if (ctl->met_np <= 0) {
03335     NC(nc_inq_varid(ncid, levname, &varid));
03336     NC(nc_get_var_double(ncid, varid, met->p));
03337     for (int ip = 0; ip < met->np; ip++)
03338         met->p[ip] /= 100.;
03339     LOG(2, "Altitude levels: %g, %g ... %g km",
03340         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03341     LOG(2, "Pressure levels: %g, %g ... %g hPa",
03342         met->p[0], met->p[1], met->p[met->np - 1]);
03343 }
03344 }
03345
03346 /*****
03347
03348 int read_met_help_3d(
03349     int ncid,
03350     char *varname,
03351     char *varname2,
03352     met_t * met,
03353     float dest[EX][EY][EP],
03354     float scl,
03355     int init) {
03356
03357     char varsel[LEN];
03358
03359     float offset, scalfac;
03360
03361     int varid;
03362
03363     /* Check if variable exists... */
03364     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03365         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03366             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03367             return 0;
03368         } else {
03369             sprintf(varsel, "%s", varname2);
03370         } else
03371             sprintf(varsel, "%s", varname);
03372
03373     /* Read packed data... */
03374     if (nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03375         && nc_get_att_float(ncid, varid, "scale_factor",
03376                             &scalfac) == NC_NOERR) {
03377
03378         /* Allocate... */
03379         short *help;
03380         ALLOC(help, short,
03381              EX * EY * EP);
03382
03383         /* Read fill value and missing value... */
03384         short fillval, missval;
03385         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03386             fillval = 0;
03387         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03388             missval = 0;
03389
03390         /* Write info... */
03391         LOG(2, "Read 3-D variable: %s "
03392             " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03393             varsel, fillval, missval, scalfac, offset);
03394
03395         /* Read data... */
03396         NC(nc_get_var_short(ncid, varid, help));
03397
03398         /* Copy and check data... */
03399 #pragma omp parallel for default(shared) num_threads(12)
03400         for (int ix = 0; ix < met->nx; ix++)
03401             for (int iy = 0; iy < met->ny; iy++)
03402                 for (int ip = 0; ip < met->np; ip++) {
03403                     if (init)

```

```

03404         dest[ix][iy][ip] = 0;
03405         short aux = help[(ip * met->ny + iy) * met->nx + ix];
03406         if ((fillval == 0 || aux != fillval)
03407             && (missval == 0 || aux != missval)
03408             && fabsf(aux * scalfac + offset) < 1e14f)
03409             dest[ix][iy][ip] += scl * (aux * scalfac + offset);
03410         else
03411             dest[ix][iy][ip] = GSL_NAN;
03412     }
03413
03414     /* Free... */
03415     free(help);
03416 }
03417
03418 /* Unpacked data... */
03419 else {
03420
03421     /* Allocate... */
03422     float *help;
03423     ALLOC(help, float,
03424           EX * EY * EP);
03425
03426     /* Read fill value and missing value... */
03427     float fillval, missval;
03428     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03429         fillval = 0;
03430     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03431         missval = 0;
03432
03433     /* Write info... */
03434     LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
03435         varsel, fillval, missval);
03436
03437     /* Read data... */
03438     NC(nc_get_var_float(ncid, varid, help));
03439
03440     /* Copy and check data... */
03441     #pragma omp parallel for default(shared) num_threads(12)
03442     for (int ix = 0; ix < met->nx; ix++)
03443         for (int iy = 0; iy < met->ny; iy++)
03444             for (int ip = 0; ip < met->np; ip++) {
03445                 if (init)
03446                     dest[ix][iy][ip] = 0;
03447                 float aux = help[(ip * met->ny + iy) * met->nx + ix];
03448                 if ((fillval == 0 || aux != fillval)
03449                     && (missval == 0 || aux != missval)
03450                     && fabsf(aux) < 1e14f)
03451                     dest[ix][iy][ip] += scl * aux;
03452                 else
03453                     dest[ix][iy][ip] = GSL_NAN;
03454             }
03455
03456     /* Free... */
03457     free(help);
03458 }
03459
03460 /* Return... */
03461 return 1;
03462 }
03463
03464 /*****
03465
03466 int read_met_help_2d(
03467     int ncid,
03468     char *varname,
03469     char *varname2,
03470     met_t *met,
03471     float dest[EX][EY],
03472     float scl,
03473     int init) {
03474
03475     char varsel[LEN];
03476
03477     float offset, scalfac;
03478
03479     int varid;
03480
03481     /* Check if variable exists... */
03482     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03483         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03484             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03485             return 0;
03486         } else {
03487             sprintf(varsel, "%s", varname2);
03488         } else
03489             sprintf(varsel, "%s", varname);
03490

```

```

03491  /* Read packed data... */
03492  if (nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03493      && nc_get_att_float(ncid, varid, "scale_factor",
03494                          &scalfac) == NC_NOERR) {
03495
03496      /* Write info... */
03497      LOG(2, "Packed: scale_factor= %g / add_offset= %g", scalfac, offset);
03498
03499      /* Allocate... */
03500      short *help;
03501      ALLOC(help, short,
03502            EX * EY * EP);
03503
03504      /* Read fill value and missing value... */
03505      short fillval, missval;
03506      if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03507          fillval = 0;
03508      if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03509          missval = 0;
03510
03511      /* Write info... */
03512      LOG(2, "Read 2-D variable: %s"
03513          " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03514          varsel, fillval, missval, scalfac, offset);
03515
03516      /* Read data... */
03517      NC(nc_get_var_short(ncid, varid, help));
03518
03519      /* Copy and check data... */
03520      #pragma omp parallel for default(shared) num_threads(12)
03521      for (int ix = 0; ix < met->nx; ix++)
03522          for (int iy = 0; iy < met->ny; iy++) {
03523              if (init)
03524                  dest[ix][iy] = 0;
03525              short aux = help[iy * met->nx + ix];
03526              if ((fillval == 0 || aux != fillval)
03527                  && (missval == 0 || aux != missval)
03528                  && fabsf(aux * scalfac + offset) < 1e14f)
03529                  dest[ix][iy] += scl * (aux * scalfac + offset);
03530              else
03531                  dest[ix][iy] = GSL_NAN;
03532          }
03533
03534      /* Free... */
03535      free(help);
03536  }
03537
03538  /* Unpacked data... */
03539  else {
03540
03541      /* Allocate... */
03542      float *help;
03543      ALLOC(help, float,
03544            EX * EY);
03545
03546      /* Read fill value and missing value... */
03547      float fillval, missval;
03548      if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03549          fillval = 0;
03550      if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03551          missval = 0;
03552
03553      /* Write info... */
03554      LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03555          varsel, fillval, missval);
03556
03557      /* Read data... */
03558      NC(nc_get_var_float(ncid, varid, help));
03559
03560      /* Copy and check data... */
03561      #pragma omp parallel for default(shared) num_threads(12)
03562      for (int ix = 0; ix < met->nx; ix++)
03563          for (int iy = 0; iy < met->ny; iy++) {
03564              if (init)
03565                  dest[ix][iy] = 0;
03566              float aux = help[iy * met->nx + ix];
03567              if ((fillval == 0 || aux != fillval)
03568                  && (missval == 0 || aux != missval)
03569                  && fabsf(aux) < 1e14f)
03570                  dest[ix][iy] += scl * aux;
03571              else
03572                  dest[ix][iy] = GSL_NAN;
03573          }
03574
03575      /* Free... */
03576      free(help);
03577  }

```

```

03578
03579  /* Return... */
03580  return 1;
03581 }
03582
03583 /*****
03584
03585 void read_met_levels(
03586     int ncid,
03587     ctl_t * ctl,
03588     met_t * met) {
03589
03590     /* Set timer... */
03591     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03592     LOG(2, "Read level data...");
03593
03594     /* Read meteorological data... */
03595     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0, 1))
03596         ERRMSG("Cannot read temperature!");
03597     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0, 1))
03598         ERRMSG("Cannot read zonal wind!");
03599     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0, 1))
03600         ERRMSG("Cannot read meridional wind!");
03601     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f, 1))
03602         WARN("Cannot read vertical velocity!");
03603     if (!read_met_help_3d
03604         (ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O), 1))
03605         WARN("Cannot read specific humidity!");
03606     if (!read_met_help_3d
03607         (ncid, "o3", "O3", met, met->o3, (float) (MA / MO3), 1))
03608         WARN("Cannot read ozone data!");
03609     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03610         if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0, 1))
03611             WARN("Cannot read cloud liquid water content!");
03612         if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0, 1))
03613             WARN("Cannot read cloud ice water content!");
03614     }
03615     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03616         if (!read_met_help_3d
03617             (ncid, "crwc", "CRWC", met, met->lwc, 1.0, ctl->met_cloud == 2))
03618             WARN("Cannot read cloud rain water content!");
03619         if (!read_met_help_3d
03620             (ncid, "cswc", "CSWC", met, met->iwc, 1.0, ctl->met_cloud == 2))
03621             WARN("Cannot read cloud snow water content!");
03622     }
03623
03624     /* Transfer from model levels to pressure levels... */
03625     if (ctl->met_np > 0) {
03626
03627         /* Read pressure on model levels... */
03628         if (!read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f, 1))
03629             ERRMSG("Cannot read pressure on model levels!");
03630
03631         /* Vertical interpolation from model to pressure levels... */
03632         read_met_ml2pl(ctl, met, met->t);
03633         read_met_ml2pl(ctl, met, met->u);
03634         read_met_ml2pl(ctl, met, met->v);
03635         read_met_ml2pl(ctl, met, met->w);
03636         read_met_ml2pl(ctl, met, met->h2o);
03637         read_met_ml2pl(ctl, met, met->o3);
03638         read_met_ml2pl(ctl, met, met->lwc);
03639         read_met_ml2pl(ctl, met, met->iwc);
03640
03641         /* Set new pressure levels... */
03642         met->np = ctl->met_np;
03643         for (int ip = 0; ip < met->np; ip++)
03644             met->p[ip] = ctl->met_p[ip];
03645     }
03646
03647     /* Check ordering of pressure levels... */
03648     for (int ip = 1; ip < met->np; ip++)
03649         if (met->p[ip - 1] < met->p[ip])
03650             ERRMSG("Pressure levels must be descending!");
03651 }
03652
03653 /*****
03654
03655 void read_met_ml2pl(
03656     ctl_t * ctl,
03657     met_t * met,
03658     float var[EX][EY][EP]) {
03659
03660     double aux[EP], p[EP];
03661
03662     /* Set timer... */
03663     SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03664     LOG(2, "Interpolate meteo data to pressure levels...");

```

```

03665
03666 /* Loop over columns... */
03667 #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03668 for (int ix = 0; ix < met->nx; ix++)
03669     for (int iy = 0; iy < met->ny; iy++) {
03670
03671         /* Copy pressure profile... */
03672         for (int ip = 0; ip < met->np; ip++)
03673             p[ip] = met->p[ix][iy][ip];
03674
03675         /* Interpolate... */
03676         for (int ip = 0; ip < ctl->met_np; ip++) {
03677             double pt = ctl->met_p[ip];
03678             if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03679                 pt = p[0];
03680             else if ((pt > p[met->np - 1] && p[1] > p[0])
03681                     || (pt < p[met->np - 1] && p[1] < p[0]))
03682                 pt = p[met->np - 1];
03683             int ip2 = locate_irr(p, met->np, pt);
03684             aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03685                          p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03686         }
03687
03688         /* Copy data... */
03689         for (int ip = 0; ip < ctl->met_np; ip++)
03690             var[ix][iy][ip] = (float) aux[ip];
03691     }
03692 }
03693
03694 /*****
03695
03696 void read_met_pbl(
03697     met_t * met) {
03698
03699     /* Set timer... */
03700     SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
03701     LOG(2, "Calculate planetary boundary layer...");
03702
03703     /* Parameters used to estimate the height of the PBL
03704        (e.g., Vogelezang and Holtslag, 1996; Seidel et al., 2012)... */
03705     const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
03706
03707     /* Loop over grid points... */
03708     #pragma omp parallel for default(shared) collapse(2)
03709     for (int ix = 0; ix < met->nx; ix++)
03710         for (int iy = 0; iy < met->ny; iy++) {
03711
03712             /* Set bottom level of PBL... */
03713             double pbl_bot = met->p[ix][iy] + DZ2DP(dz, met->p[ix][iy]);
03714
03715             /* Find lowest level near the bottom... */
03716             int ip;
03717             for (ip = 1; ip < met->np; ip++)
03718                 if (met->p[ip] < pbl_bot)
03719                     break;
03720
03721             /* Get near surface data... */
03722             double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
03723                           met->p[ip], met->z[ix][iy][ip], pbl_bot);
03724             double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
03725                           met->p[ip], met->t[ix][iy][ip], pbl_bot);
03726             double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
03727                           met->p[ip], met->u[ix][iy][ip], pbl_bot);
03728             double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
03729                           met->p[ip], met->v[ix][iy][ip], pbl_bot);
03730             double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
03731                              met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
03732             double tvs = THETA_VIRT(pbl_bot, ts, h2os);
03733
03734             /* Init... */
03735             double rib, rib_old = 0;
03736
03737             /* Loop over levels... */
03738             for (; ip < met->np; ip++) {
03739
03740                 /* Get squared horizontal wind speed... */
03741                 double vh2
03742                     = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
03743                 vh2 = GSL_MAX(vh2, SQR(umin));
03744
03745                 /* Calculate bulk Richardson number... */
03746                 rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
03747                     * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
03748                                   met->h2o[ix][iy][ip]) - tvs) / vh2;
03749
03750                 /* Check for critical value... */
03751                 if (rib >= rib_crit) {

```

```

03752         met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
03753                                     rib, met->p[ip], rib_crit));
03754         if (met->pbl[ix][iy] > pbl_bot)
03755             met->pbl[ix][iy] = (float) pbl_bot;
03756         break;
03757     }
03758
03759     /* Save Richardson number... */
03760     rib_old = rib;
03761 }
03762 }
03763 }
03764
03765 /*****
03766
03767 void read_met_periodic(
03768     met_t * met) {
03769
03770     /* Set timer... */
03771     SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
03772     LOG(2, "Apply periodic boundary conditions...");
03773
03774     /* Check longitudes... */
03775     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
03776               + met->lon[1] - met->lon[0] - 360) < 0.01))
03777         return;
03778
03779     /* Increase longitude counter... */
03780     if ((++met->nx) > EX)
03781         ERRMSG("Cannot create periodic boundary conditions!");
03782
03783     /* Set longitude... */
03784     met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
03785
03786     /* Loop over latitudes and pressure levels... */
03787     #pragma omp parallel for default(shared)
03788     for (int iy = 0; iy < met->ny; iy++) {
03789         met->ps[met->nx - 1][iy] = met->ps[0][iy];
03790         met->zs[met->nx - 1][iy] = met->zs[0][iy];
03791         met->ts[met->nx - 1][iy] = met->ts[0][iy];
03792         met->us[met->nx - 1][iy] = met->us[0][iy];
03793         met->vs[met->nx - 1][iy] = met->vs[0][iy];
03794         for (int ip = 0; ip < met->np; ip++) {
03795             met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
03796             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
03797             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
03798             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
03799             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
03800             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
03801             met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
03802             met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
03803         }
03804     }
03805 }
03806
03807 /*****
03808
03809 void read_met_pv(
03810     met_t * met) {
03811
03812     double pows[EP];
03813
03814     /* Set timer... */
03815     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
03816     LOG(2, "Calculate potential vorticity...");
03817
03818     /* Set powers... */
03819     #pragma omp parallel for default(shared)
03820     for (int ip = 0; ip < met->np; ip++)
03821         pows[ip] = pow(1000. / met->p[ip], 0.286);
03822
03823     /* Loop over grid points... */
03824     #pragma omp parallel for default(shared)
03825     for (int ix = 0; ix < met->nx; ix++) {
03826
03827         /* Set indices... */
03828         int ix0 = GSL_MAX(ix - 1, 0);
03829         int ix1 = GSL_MIN(ix + 1, met->nx - 1);
03830
03831         /* Loop over grid points... */
03832         for (int iy = 0; iy < met->ny; iy++) {
03833
03834             /* Set indices... */
03835             int iy0 = GSL_MAX(iy - 1, 0);
03836             int iy1 = GSL_MIN(iy + 1, met->ny - 1);
03837
03838             /* Set auxiliary variables... */

```

```

03839     double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
03840     double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
03841     double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
03842     double c0 = cos(met->lat[iy0] / 180. * M_PI);
03843     double c1 = cos(met->lat[iy1] / 180. * M_PI);
03844     double cr = cos(latr / 180. * M_PI);
03845     double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
03846
03847     /* Loop over grid points... */
03848     for (int ip = 0; ip < met->np; ip++) {
03849
03850         /* Get gradients in longitude... */
03851         double dtdx
03852             = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
03853         double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
03854
03855         /* Get gradients in latitude... */
03856         double dtdy
03857             = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03858         double dudy
03859             = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03860
03861         /* Set indices... */
03862         int ip0 = GSL_MAX(ip - 1, 0);
03863         int ip1 = GSL_MIN(ip + 1, met->np - 1);
03864
03865         /* Get gradients in pressure... */
03866         double dtdp, dudp, dvdp;
03867         double dp0 = 100. * (met->p[ip] - met->p[ip0]);
03868         double dp1 = 100. * (met->p[ip1] - met->p[ip]);
03869         if (ip != ip0 && ip != ip1) {
03870             double denom = dp0 * dp1 * (dp0 + dp1);
03871             dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03872                 - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03873                 + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03874                 / denom;
03875             dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03876                 - dp1 * dp1 * met->u[ix][iy][ip0]
03877                 + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03878                 / denom;
03879             dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03880                 - dp1 * dp1 * met->v[ix][iy][ip0]
03881                 + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03882                 / denom;
03883         } else {
03884             double denom = dp0 + dp1;
03885             dtdp =
03886                 (met->t[ix][iy][ip1] * pows[ip1] -
03887                 met->t[ix][iy][ip0] * pows[ip0]) / denom;
03888             dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03889             dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03890         }
03891
03892         /* Calculate PV... */
03893         met->pv[ix][iy][ip] = (float)
03894             (1e6 * G0 *
03895             (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03896     }
03897 }
03898 }
03899
03900 /* Fix for polar regions... */
03901 #pragma omp parallel for default(shared)
03902 for (int ix = 0; ix < met->nx; ix++)
03903     for (int ip = 0; ip < met->np; ip++) {
03904         met->pv[ix][0][ip]
03905             = met->pv[ix][1][ip]
03906             = met->pv[ix][2][ip];
03907         met->pv[ix][met->ny - 1][ip]
03908             = met->pv[ix][met->ny - 2][ip]
03909             = met->pv[ix][met->ny - 3][ip];
03910     }
03911 }
03912
03913 /*****
03914
03915 void read_met_sample(
03916     ctl_t * ctl,
03917     met_t * met) {
03918
03919     met_t *help;
03920
03921     /* Check parameters... */
03922     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03923         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03924         return;
03925

```



```

03926  /* Set timer... */
03927  SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
03928  LOG(2, "Downsampling of meteo data...");
03929
03930  /* Allocate... */
03931  ALLOC(help, met_t, 1);
03932
03933  /* Copy data... */
03934  help->nx = met->nx;
03935  help->ny = met->ny;
03936  help->np = met->np;
03937  memcpy(help->lon, met->lon, sizeof(met->lon));
03938  memcpy(help->lat, met->lat, sizeof(met->lat));
03939  memcpy(help->p, met->p, sizeof(met->p));
03940
03941  /* Smoothing... */
03942  for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
03943      for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
03944          for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
03945              help->ps[ix][iy] = 0;
03946              help->zs[ix][iy] = 0;
03947              help->ts[ix][iy] = 0;
03948              help->us[ix][iy] = 0;
03949              help->vs[ix][iy] = 0;
03950              help->t[ix][iy][ip] = 0;
03951              help->u[ix][iy][ip] = 0;
03952              help->v[ix][iy][ip] = 0;
03953              help->w[ix][iy][ip] = 0;
03954              help->h2o[ix][iy][ip] = 0;
03955              help->o3[ix][iy][ip] = 0;
03956              help->lwc[ix][iy][ip] = 0;
03957              help->iwc[ix][iy][ip] = 0;
03958              float wsum = 0;
03959              for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
03960                  ix2++) {
03961                  int ix3 = ix2;
03962                  if (ix3 < 0)
03963                      ix3 += met->nx;
03964                  else if (ix3 >= met->nx)
03965                      ix3 -= met->nx;
03966
03967                  for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03968                      iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03969                      for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03970                          ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03971                          float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
03972                              * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
03973                              * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
03974                          help->ps[ix][iy] += w * met->ps[ix3][iy2];
03975                          help->zs[ix][iy] += w * met->zs[ix3][iy2];
03976                          help->ts[ix][iy] += w * met->ts[ix3][iy2];
03977                          help->us[ix][iy] += w * met->us[ix3][iy2];
03978                          help->vs[ix][iy] += w * met->vs[ix3][iy2];
03979                          help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03980                          help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03981                          help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03982                          help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03983                          help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03984                          help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03985                          help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03986                          help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03987                          wsum += w;
03988                      }
03989                  }
03990              help->ps[ix][iy] /= wsum;
03991              help->zs[ix][iy] /= wsum;
03992              help->ts[ix][iy] /= wsum;
03993              help->us[ix][iy] /= wsum;
03994              help->vs[ix][iy] /= wsum;
03995              help->t[ix][iy][ip] /= wsum;
03996              help->u[ix][iy][ip] /= wsum;
03997              help->v[ix][iy][ip] /= wsum;
03998              help->w[ix][iy][ip] /= wsum;
03999              help->h2o[ix][iy][ip] /= wsum;
04000              help->o3[ix][iy][ip] /= wsum;
04001              help->lwc[ix][iy][ip] /= wsum;
04002              help->iwc[ix][iy][ip] /= wsum;
04003          }
04004      }
04005  }
04006
04007  /* Downsampling... */
04008  met->nx = 0;
04009  for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04010      met->lon[met->nx] = help->lon[ix];
04011      met->ny = 0;
04012      for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {

```

```

04013     met->lat[met->ny] = help->lat[iy];
04014     met->ps[met->nx][met->ny] = help->ps[ix][iy];
04015     met->zs[met->nx][met->ny] = help->zs[ix][iy];
04016     met->ts[met->nx][met->ny] = help->ts[ix][iy];
04017     met->us[met->nx][met->ny] = help->us[ix][iy];
04018     met->vs[met->nx][met->ny] = help->vs[ix][iy];
04019     met->np = 0;
04020     for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04021         met->p[met->np] = help->p[ip];
04022         met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04023         met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04024         met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04025         met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04026         met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04027         met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04028         met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
04029         met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04030         met->np++;
04031     }
04032     met->ny++;
04033 }
04034 met->nx++;
04035 }
04036
04037 /* Free... */
04038 free(help);
04039 }
04040
04041 /*****
04042
04043 void read_met_surface(
04044     int ncid,
04045     met_t * met) {
04046
04047     /* Set timer... */
04048     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04049     LOG(2, "Read surface data...");
04050
04051     /* Read surface pressure... */
04052     if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0f, 1)) {
04053         if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f, 1)) {
04054             WARN("Cannot not read surface pressure data (use lowest level!)");
04055             for (int ix = 0; ix < met->nx; ix++)
04056                 for (int iy = 0; iy < met->ny; iy++)
04057                     met->ps[ix][iy] = (float) met->p[0];
04058         }
04059     } else
04060         for (int ix = 0; ix < met->nx; ix++)
04061             for (int iy = 0; iy < met->ny; iy++)
04062                 met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04063
04064     /* Read geopotential height at the surface... */
04065     if (!read_met_help_2d
04066         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0)), 1))
04067         if (!read_met_help_2d
04068             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.), 1))
04069             WARN("Cannot read surface geopotential height!");
04070
04071     /* Read temperature at the surface... */
04072     if (!read_met_help_2d(ncid, "t2m", "T2M", met, met->ts, 1.0, 1))
04073         WARN("Cannot read surface temperature!");
04074
04075     /* Read zonal wind at the surface... */
04076     if (!read_met_help_2d(ncid, "u10m", "U10M", met, met->us, 1.0, 1))
04077         WARN("Cannot read surface zonal wind!");
04078
04079     /* Read meridional wind at the surface... */
04080     if (!read_met_help_2d(ncid, "v10m", "V10M", met, met->vs, 1.0, 1))
04081         WARN("Cannot read surface meridional wind!");
04082 }
04083
04084 /*****
04085
04086 void read_met_tropo(
04087     ctl_t * ctl,
04088     met_t * met) {
04089
04090     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04091         th2[200], z[EP], z2[200];
04092
04093     /* Set timer... */
04094     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04095     LOG(2, "Calculate tropopause...");
04096
04097     /* Get altitude and pressure profiles... */
04098     #pragma omp parallel for default(shared)
04099     for (int iz = 0; iz < met->np; iz++)

```

```

04100     z[iz] = Z(met->p[iz]);
04101 #pragma omp parallel for default(shared)
04102     for (int iz = 0; iz <= 190; iz++) {
04103         z2[iz] = 4.5 + 0.1 * iz;
04104         p2[iz] = P(z2[iz]);
04105     }
04106
04107     /* Do not calculate tropopause... */
04108     if (ctl->met_tropo == 0)
04109 #pragma omp parallel for default(shared) collapse(2)
04110         for (int ix = 0; ix < met->nx; ix++)
04111             for (int iy = 0; iy < met->ny; iy++)
04112                 met->pt[ix][iy] = GSL_NAN;
04113
04114     /* Use tropopause climatology... */
04115     else if (ctl->met_tropo == 1) {
04116 #pragma omp parallel for default(shared) collapse(2)
04117         for (int ix = 0; ix < met->nx; ix++)
04118             for (int iy = 0; iy < met->ny; iy++)
04119                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
04120     }
04121
04122     /* Use cold point... */
04123     else if (ctl->met_tropo == 2) {
04124
04125         /* Loop over grid points... */
04126 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04127         for (int ix = 0; ix < met->nx; ix++)
04128             for (int iy = 0; iy < met->ny; iy++) {
04129
04130                 /* Interpolate temperature profile... */
04131                 for (int iz = 0; iz < met->np; iz++)
04132                     t[iz] = met->t[ix][iy][iz];
04133                 spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04134
04135                 /* Find minimum... */
04136                 int iz = (int) gsl_stats_min_index(t2, 1, 171);
04137                 if (iz > 0 && iz < 170)
04138                     met->pt[ix][iy] = (float) p2[iz];
04139                 else
04140                     met->pt[ix][iy] = GSL_NAN;
04141             }
04142     }
04143
04144     /* Use WMO definition... */
04145     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04146
04147         /* Loop over grid points... */
04148 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04149         for (int ix = 0; ix < met->nx; ix++)
04150             for (int iy = 0; iy < met->ny; iy++) {
04151
04152                 /* Interpolate temperature profile... */
04153                 int iz;
04154                 for (iz = 0; iz < met->np; iz++)
04155                     t[iz] = met->t[ix][iy][iz];
04156                 spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04157
04158                 /* Find 1st tropopause... */
04159                 met->pt[ix][iy] = GSL_NAN;
04160                 for (iz = 0; iz <= 170; iz++) {
04161                     int found = 1;
04162                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04163                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04164                             ctl->met_tropo_lapse) {
04165                             found = 0;
04166                             break;
04167                         }
04168                     if (found) {
04169                         if (iz > 0 && iz < 170)
04170                             met->pt[ix][iy] = (float) p2[iz];
04171                         break;
04172                     }
04173                 }
04174
04175                 /* Find 2nd tropopause... */
04176                 if (ctl->met_tropo == 4) {
04177                     met->pt[ix][iy] = GSL_NAN;
04178                     for (; iz <= 170; iz++) {
04179                         int found = 1;
04180                         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04181                             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04182                                 ctl->met_tropo_lapse_sep) {
04183                                 found = 0;
04184                                 break;
04185                             }
04186                     if (found)

```

```

04187         break;
04188     }
04189     for (; iz <= 170; iz++) {
04190         int found = 1;
04191         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04192             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04193                 ctl->met_tropo_lapse) {
04194                 found = 0;
04195                 break;
04196             }
04197         if (found) {
04198             if (iz > 0 && iz < 170)
04199                 met->pt[ix][iy] = (float) p2[iz];
04200             break;
04201         }
04202     }
04203 }
04204 }
04205 }
04206
04207 /* Use dynamical tropopause... */
04208 else if (ctl->met_tropo == 5) {
04209     /* Loop over grid points... */
04210     #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04211     for (int ix = 0; ix < met->nx; ix++)
04212         for (int iy = 0; iy < met->ny; iy++) {
04213
04214             /* Interpolate potential vorticity profile... */
04215             for (int iz = 0; iz < met->np; iz++)
04216                 pv[iz] = met->pv[ix][iy][iz];
04217             spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04218
04219             /* Interpolate potential temperature profile... */
04220             for (int iz = 0; iz < met->np; iz++)
04221                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04222             spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04223
04224             /* Find dynamical tropopause... */
04225             met->pt[ix][iy] = GSL_NAN;
04226             for (int iz = 0; iz <= 170; iz++)
04227                 if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04228                     || th2[iz] >= ctl->met_tropo_theta) {
04229                     if (iz > 0 && iz < 170)
04230                         met->pt[ix][iy] = (float) p2[iz];
04231                     break;
04232                 }
04233         }
04234     }
04235 }
04236
04237 else
04238     ERRMSG("Cannot calculate tropopause!");
04239
04240 /* Interpolate temperature, geopotential height, and water vapor vmr... */
04241 #pragma omp parallel for default(shared) collapse(2)
04242 for (int ix = 0; ix < met->nx; ix++)
04243     for (int iy = 0; iy < met->ny; iy++) {
04244         double h2ot, tt, zt;
04245         INTPOL_INIT;
04246         intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04247                             met->lat[iy], &tt, ci, cw, 1);
04248         intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04249                             met->lat[iy], &zt, ci, cw, 0);
04250         intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04251                             met->lat[iy], &h2ot, ci, cw, 0);
04252         met->tt[ix][iy] = (float) tt;
04253         met->zt[ix][iy] = (float) zt;
04254         met->h2ot[ix][iy] = (float) h2ot;
04255     }
04256 }
04257
04258 /*****
04259
04260 double scan_ctl(
04261     const char *filename,
04262     int argc,
04263     char *argv[],
04264     const char *varname,
04265     int arid,
04266     const char *defvalue,
04267     char *value) {
04268
04269     FILE *in = NULL;
04270
04271     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
04272           rvarname[LEN], rval[LEN];
04273

```

```

04274     int contain = 0, i;
04275
04276     /* Open file... */
04277     if (filename[strlen(filename) - 1] != '-')
04278         if (!(in = fopen(filename, "r")))
04279             ERRMSG("Cannot open file!");
04280
04281     /* Set full variable name... */
04282     if (arridx >= 0) {
04283         sprintf(fullname1, "%s[%d]", varname, arridx);
04284         sprintf(fullname2, "%s[*]", varname);
04285     } else {
04286         sprintf(fullname1, "%s", varname);
04287         sprintf(fullname2, "%s", varname);
04288     }
04289
04290     /* Read data... */
04291     if (in != NULL)
04292         while (fgets(line, LEN, in))
04293             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
04294                 if (strcasecmp(rvarname, fullname1) == 0 ||
04295                     strcasecmp(rvarname, fullname2) == 0) {
04296                     contain = 1;
04297                     break;
04298                 }
04299     for (i = 1; i < argc - 1; i++)
04300         if (strcasecmp(argv[i], fullname1) == 0 ||
04301             strcasecmp(argv[i], fullname2) == 0) {
04302             sprintf(rval, "%s", argv[i + 1]);
04303             contain = 1;
04304             break;
04305         }
04306
04307     /* Close file... */
04308     if (in != NULL)
04309         fclose(in);
04310
04311     /* Check for missing variables... */
04312     if (!contain) {
04313         if (strlen(defvalue) > 0)
04314             sprintf(rval, "%s", defvalue);
04315         else
04316             ERRMSG("Missing variable %s!\n", fullname1);
04317     }
04318
04319     /* Write info... */
04320     LOG(1, "%s = %s", fullname1, rval);
04321
04322     /* Return values... */
04323     if (value != NULL)
04324         sprintf(value, "%s", rval);
04325     return atof(rval);
04326 }
04327
04328 /*****
04329
04330 double sedi(
04331     double p,
04332     double T,
04333     double r_p,
04334     double rho_p) {
04335
04336     double eta, G, K, lambda, rho, v;
04337
04338     /* Convert pressure from hPa to Pa... */
04339     p *= 100.;
04340
04341     /* Convert particle radius from microns to m... */
04342     r_p *= 1e-6;
04343
04344     /* Density of dry air [kg / m^3]... */
04345     rho = p / (RA * T);
04346
04347     /* Dynamic viscosity of air [kg / (m s)]... */
04348     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04349
04350     /* Thermal velocity of an air molecule [m / s]... */
04351     v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04352
04353     /* Mean free path of an air molecule [m]... */
04354     lambda = 2. * eta / (rho * v);
04355
04356     /* Knudsen number for air (dimensionless)... */
04357     K = lambda / r_p;
04358
04359     /* Cunningham slip-flow correction (dimensionless)... */
04360     G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));

```

```

04361
04362  /* Sedimentation velocity [m / s]... */
04363  return 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
04364 }
04365
04366 /*****
04367
04368 void spline(
04369     double *x,
04370     double *y,
04371     int n,
04372     double *x2,
04373     double *y2,
04374     int n2,
04375     int method) {
04376
04377     /* Cubic spline interpolation... */
04378     if (method == 1) {
04379
04380         /* Allocate... */
04381         gsl_interp_accel *acc;
04382         gsl_spline *s;
04383         acc = gsl_interp_accel_alloc();
04384         s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04385
04386         /* Interpolate profile... */
04387         gsl_spline_init(s, x, y, (size_t) n);
04388         for (int i = 0; i < n2; i++)
04389             if (x2[i] <= x[0])
04390                 y2[i] = y[0];
04391             else if (x2[i] >= x[n - 1])
04392                 y2[i] = y[n - 1];
04393             else
04394                 y2[i] = gsl_spline_eval(s, x2[i], acc);
04395
04396         /* Free... */
04397         gsl_spline_free(s);
04398         gsl_interp_accel_free(acc);
04399     }
04400
04401     /* Linear interpolation... */
04402     else {
04403         for (int i = 0; i < n2; i++)
04404             if (x2[i] <= x[0])
04405                 y2[i] = y[0];
04406             else if (x2[i] >= x[n - 1])
04407                 y2[i] = y[n - 1];
04408             else {
04409                 int idx = locate_irr(x, n, x2[i]);
04410                 y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04411             }
04412     }
04413 }
04414
04415 /*****
04416
04417 float stddev(
04418     float *data,
04419     int n) {
04420
04421     if (n <= 0)
04422         return 0;
04423
04424     float mean = 0, var = 0;
04425
04426     for (int i = 0; i < n; ++i) {
04427         mean += data[i];
04428         var += SQR(data[i]);
04429     }
04430
04431     return sqrtf(var / (float) n - SQR(mean / (float) n));
04432 }
04433
04434 /*****
04435
04436 void time2jsec(
04437     int year,
04438     int mon,
04439     int day,
04440     int hour,
04441     int min,
04442     int sec,
04443     double remain,
04444     double *jsec) {
04445
04446     struct tm t0, t1;
04447

```

```

04448     t0.tm_year = 100;
04449     t0.tm_mon = 0;
04450     t0.tm_mday = 1;
04451     t0.tm_hour = 0;
04452     t0.tm_min = 0;
04453     t0.tm_sec = 0;
04454
04455     t1.tm_year = year - 1900;
04456     t1.tm_mon = mon - 1;
04457     t1.tm_mday = day;
04458     t1.tm_hour = hour;
04459     t1.tm_min = min;
04460     t1.tm_sec = sec;
04461
04462     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04463 }
04464
04465 /*****
04466
04467 void timer(
04468     const char *name,
04469     const char *group,
04470     int output) {
04471
04472     static char names[NTIMER][100], groups[NTIMER][100];
04473
04474     static double rt_name[NTIMER], rt_group[NTIMER], t0, t1;
04475
04476     static int iname = -1, igroup = -1, nname, ngroup;
04477
04478     /* Get time... */
04479     t1 = omp_get_wtime();
04480
04481     /* Add elapsed time to current timers... */
04482     if (iname >= 0)
04483         rt_name[iname] += t1 - t0;
04484     if (igroup >= 0)
04485         rt_group[igroup] += t1 - t0;
04486
04487     /* Report timers... */
04488     if (output) {
04489         for (int i = 0; i < nname; i++)
04490             LOG(1, "TIMER_%s = %.3f s", names[i], rt_name[i]);
04491         for (int i = 0; i < ngroup; i++)
04492             LOG(1, "TIMER_%s = %.3f s", groups[i], rt_group[i]);
04493         double total = 0.0;
04494         for (int i = 0; i < nname; i++)
04495             total += rt_name[i];
04496         LOG(1, "TIMER_TOTAL = %.3f s", total);
04497     }
04498
04499     /* Identify IDs of next timer... */
04500     for (iname = 0; iname < nname; iname++)
04501         if (strcasecmp(name, names[iname]) == 0)
04502             break;
04503     for (igroup = 0; igroup < ngroup; igroup++)
04504         if (strcasecmp(group, groups[igroup]) == 0)
04505             break;
04506
04507     /* Check whether this is a new timer... */
04508     if (iname >= nname) {
04509         sprintf(names[iname], "%s", name);
04510         if ((++nname) > NTIMER)
04511             ERRMSG("Too many timers!");
04512     }
04513
04514     /* Check whether this is a new group... */
04515     if (igroup >= ngroup) {
04516         sprintf(groups[igroup], "%s", group);
04517         if ((++ngroup) > NTIMER)
04518             ERRMSG("Too many groups!");
04519     }
04520
04521     /* Save starting time... */
04522     t0 = t1;
04523 }
04524
04525 /*****
04526
04527 double tropo_weight(
04528     double t,
04529     double lat,
04530     double p) {
04531
04532     /* Get tropopause pressure... */
04533     double pt = clim_tropo(t, lat);
04534

```

```

04535  /* Get pressure range... */
04536  double p1 = pt * 0.866877899;
04537  double p0 = pt / 0.866877899;
04538
04539  /* Get weighting factor... */
04540  if (p > p0)
04541      return 1;
04542  else if (p < p1)
04543      return 0;
04544  else
04545      return LIN(p0, 1.0, p1, 0.0, p);
04546 }
04547
04548 /*****
04549
04550 void write_atm(
04551     const char *filename,
04552     ctl_t * ctl,
04553     atm_t * atm,
04554     double t) {
04555
04556     FILE *in, *out;
04557
04558     char line[LEN];
04559
04560     double r, t0, t1;
04561
04562     int year, mon, day, hour, min, sec;
04563
04564     /* Set timer... */
04565     SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
04566
04567     /* Set time interval for output... */
04568     t0 = t - 0.5 * ctl->dt_mod;
04569     t1 = t + 0.5 * ctl->dt_mod;
04570
04571     /* Write info... */
04572     LOG(1, "Write atmospheric data: %s", filename);
04573
04574     /* Write ASCII data... */
04575     if (ctl->atm_type == 0) {
04576
04577         /* Check if gnuplot output is requested... */
04578         if (ctl->atm_gpfile[0] != '-') {
04579
04580             /* Create gnuplot pipe... */
04581             if (!(out = popen("gnuplot", "w")))
04582                 ERRMSG("Cannot create pipe to gnuplot!");
04583
04584             /* Set plot filename... */
04585             fprintf(out, "set out \"%s.png\"\n", filename);
04586
04587             /* Set time string... */
04588             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04589             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04590                 year, mon, day, hour, min);
04591
04592             /* Dump gnuplot file to pipe... */
04593             if (!(in = fopen(ctl->atm_gpfile, "r")))
04594                 ERRMSG("Cannot open file!");
04595             while (fgets(line, LEN, in))
04596                 fprintf(out, "%s", line);
04597             fclose(in);
04598         }
04599     }
04600     else {
04601
04602         /* Create file... */
04603         if (!(out = fopen(filename, "w")))
04604             ERRMSG("Cannot create file!");
04605     }
04606
04607     /* Write header... */
04608     fprintf(out,
04609         "# $1 = time [s]\n"
04610         "# $2 = altitude [km]\n"
04611         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04612     for (int iq = 0; iq < ctl->nq; iq++)
04613         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
04614             ctl->qnt_unit[iq]);
04615     fprintf(out, "\n");
04616
04617     /* Write data... */
04618     for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
04619
04620         /* Check time... */
04621         if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))

```



```

04622         continue;
04623
04624         /* Write output... */
04625         fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
04626             atm->lon[ip], atm->lat[ip]);
04627         for (int iq = 0; iq < ctl->nq; iq++) {
04628             fprintf(out, " ");
04629             if (ctl->atm_filter == 1
04630                 && (atm->time[ip] < t0 || atm->time[ip] > t1))
04631                 fprintf(out, ctl->qnt_format[iq], GSL_NAN);
04632             else
04633                 fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04634         }
04635         fprintf(out, "\n");
04636     }
04637
04638     /* Close file... */
04639     fclose(out);
04640 }
04641
04642 /* Write binary data... */
04643 else if (ctl->atm_type == 1) {
04644
04645     /* Create file... */
04646     if (!(out = fopen(filename, "w")))
04647         ERRMSG("Cannot create file!");
04648
04649     /* Write data... */
04650     FWRITE(&atm->np, int,
04651         1,
04652         out);
04653     FWRITE(atm->time, double,
04654         (size_t) atm->np,
04655         out);
04656     FWRITE(atm->p, double,
04657         (size_t) atm->np,
04658         out);
04659     FWRITE(atm->lon, double,
04660         (size_t) atm->np,
04661         out);
04662     FWRITE(atm->lat, double,
04663         (size_t) atm->np,
04664         out);
04665     for (int iq = 0; iq < ctl->nq; iq++)
04666         FWRITE(atm->q[iq], double,
04667             (size_t) atm->np,
04668             out);
04669
04670     /* Close file... */
04671     fclose(out);
04672 }
04673
04674 /* Error... */
04675 else
04676     ERRMSG("Atmospheric data type not supported!");
04677
04678 /* Write info... */
04679 double mini, maxi;
04680 LOG(2, "Number of particles: %d", atm->np);
04681 gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
04682 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04683 gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
04684 LOG(2, "Altitude range: %g ... %g km", Z(mini), Z(maxi));
04685 LOG(2, "Pressure range: %g ... %g hPa", mini, maxi);
04686 gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
04687 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04688 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
04689 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04690 for (int iq = 0; iq < ctl->nq; iq++) {
04691     char msg[LEN];
04692     sprintf(msg, "Quantity %s range: %s ... %s %s",
04693         ctl->qnt_name[iq], ctl->qnt_format[iq],
04694         ctl->qnt_format[iq], ctl->qnt_unit[iq]);
04695     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
04696     LOG(2, msg, mini, maxi);
04697 }
04698 }
04699
04700 /*****
04701
04702 void write_csi(
04703     const char *filename,
04704     ctl_t * ctl,
04705     atm_t * atm,
04706     double t) {
04707
04708     static FILE *in, *out;

```

```

04709
04710 static char line[LEN];
04711
04712 static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ], rt, rt_old,
04713     rz, rlon, rlat, robs, t0, t1, area[GY], dlon, dlat, dz, lat,
04714     x[1000000], y[1000000], work[2000000];
04715
04716 static int obscount[GX][GY][GZ], ct, cx, cy, cz, ip, ix, iy, iz, n;
04717
04718 /* Set timer... */
04719 SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
04720
04721 /* Init... */
04722 if (t == ctl->t_start) {
04723
04724     /* Check quantity index for mass... */
04725     if (ctl->qnt_m < 0)
04726         ERRMSG("Need quantity mass!");
04727
04728     /* Open observation data file... */
04729     LOG(l, "Read CSI observation data: %s", ctl->csi_obsfile);
04730     if (!(in = fopen(ctl->csi_obsfile, "r")))
04731         ERRMSG("Cannot open file!");
04732
04733     /* Initialize time for file input... */
04734     rt_old = -1e99;
04735
04736     /* Create new file... */
04737     LOG(l, "Write CSI data: %s", filename);
04738     if (!(out = fopen(filename, "w")))
04739         ERRMSG("Cannot create file!");
04740
04741     /* Write header... */
04742     fprintf(out,
04743         "# $1 = time [s]\n"
04744         "# $2 = number of hits (cx)\n"
04745         "# $3 = number of misses (cy)\n"
04746         "# $4 = number of false alarms (cz)\n"
04747         "# $5 = number of observations (cx + cy)\n"
04748         "# $6 = number of forecasts (cx + cz)\n"
04749         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
04750         "# $8 = probability of detection (POD) [%%]\n"
04751         "# $9 = false alarm rate (FAR) [%%]\n"
04752         "# $10 = critical success index (CSI) [%%]\n");
04753     fprintf(out,
04754         "# $11 = hits associated with random chance\n"
04755         "# $12 = equitable threat score (ETS) [%%]\n"
04756         "# $13 = Pearson linear correlation coefficient\n"
04757         "# $14 = Spearman rank-order correlation coefficient\n"
04758         "# $15 = column density mean error (F - O) [kg/m^2]\n"
04759         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
04760         "# $17 = column density mean absolute error [kg/m^2]\n"
04761         "# $18 = number of data points\n");
04762
04763     /* Set grid box size... */
04764     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
04765     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
04766     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
04767
04768     /* Set horizontal coordinates... */
04769     for (iy = 0; iy < ctl->csi_ny; iy++) {
04770         lat = ctl->csi_lat0 + dlat * (iy + 0.5);
04771         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
04772     }
04773 }
04774
04775 /* Set time interval... */
04776 t0 = t - 0.5 * ctl->dt_mod;
04777 t1 = t + 0.5 * ctl->dt_mod;
04778
04779 /* Initialize grid cells... */
04780 #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(3)
04781 for (ix = 0; ix < ctl->csi_nx; ix++)
04782     for (iy = 0; iy < ctl->csi_ny; iy++)
04783         for (iz = 0; iz < ctl->csi_nz; iz++)
04784             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
04785
04786 /* Read observation data... */
04787 while (fgets(line, LEN, in)) {
04788
04789     /* Read data... */
04790     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
04791         5)
04792         continue;
04793
04794     /* Check time... */
04795     if (rt < t0)

```

```

04796     continue;
04797     if (rt > t1)
04798         break;
04799     if (rt < rt_old)
04800         ERRMSG("Time must be ascending!");
04801     rt_old = rt;
04802
04803     /* Check observation data... */
04804     if (!isfinite(robs))
04805         continue;
04806
04807     /* Calculate indices... */
04808     ix = (int) ((rlon - ctl->csi_lon0) / dlon);
04809     iy = (int) ((rlat - ctl->csi_lat0) / dlat);
04810     iz = (int) ((rz - ctl->csi_z0) / dz);
04811
04812     /* Check indices... */
04813     if (ix < 0 || ix >= ctl->csi_nx ||
04814         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04815         continue;
04816
04817     /* Get mean observation index... */
04818     obsmean[ix][iy][iz] += robs;
04819     obscount[ix][iy][iz]++;
04820 }
04821
04822 /* Analyze model data... */
04823 for (ip = 0; ip < atm->np; ip++) {
04824
04825     /* Check time... */
04826     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04827         continue;
04828
04829     /* Get indices... */
04830     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
04831     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
04832     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
04833
04834     /* Check indices... */
04835     if (ix < 0 || ix >= ctl->csi_nx ||
04836         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04837         continue;
04838
04839     /* Get total mass in grid cell... */
04840     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04841 }
04842
04843 /* Analyze all grid cells... */
04844 for (ix = 0; ix < ctl->csi_nx; ix++)
04845     for (iy = 0; iy < ctl->csi_ny; iy++)
04846         for (iz = 0; iz < ctl->csi_nz; iz++) {
04847
04848             /* Calculate mean observation index... */
04849             if (obscount[ix][iy][iz] > 0)
04850                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
04851
04852             /* Calculate column density... */
04853             if (modmean[ix][iy][iz] > 0)
04854                 modmean[ix][iy][iz] /= (1e6 * area[iy]);
04855
04856             /* Calculate CSI... */
04857             if (obscount[ix][iy][iz] > 0) {
04858                 ct++;
04859                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04860                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04861                     cx++;
04862                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04863                     modmean[ix][iy][iz] < ctl->csi_modmin)
04864                     cy++;
04865                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
04866                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04867                     cz++;
04868             }
04869
04870             /* Save data for other verification statistics... */
04871             if (obscount[ix][iy][iz] > 0
04872                 && (obsmean[ix][iy][iz] >= ctl->csi_obsmin
04873                     || modmean[ix][iy][iz] >= ctl->csi_modmin)) {
04874                 x[n] = modmean[ix][iy][iz];
04875                 y[n] = obsmean[ix][iy][iz];
04876                 if (++n > 1000000)
04877                     ERRMSG("Too many data points to calculate statistics!");
04878             }
04879         }
04880
04881     /* Write output... */
04882     if (fmod(t, ctl->csi_dt_out) == 0) {

```

```

04883
04884 /* Calculate verification statistics
04885 (https://www.cawcr.gov.au/projects/verification/) ... */
04886 int nobs = cx + cy;
04887 int nfor = cx + cz;
04888 double bias = (nobs > 0) ? 100. * nfor / nobs : GSL_NAN;
04889 double pod = (nobs > 0) ? (100. * cx) / nobs : GSL_NAN;
04890 double far = (nfor > 0) ? (100. * cz) / nfor : GSL_NAN;
04891 double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
04892 double cx_rd = (ct > 0) ? (1. * nobs * nfor) / ct : GSL_NAN;
04893 double ets = (cx + cy + cz - cx_rd > 0) ?
04894     (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
04895 double rho_p =
04896     (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
04897 double rho_s =
04898     (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
04899 for (int i = 0; i < n; i++)
04900     work[i] = x[i] - y[i];
04901 double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
04902 double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
04903     0.0) : GSL_NAN;
04904 double absdev =
04905     (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
04906
04907 /* Write... */
04908 fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %d\n",
04909     t, cx, cy, cz, nobs, nfor, bias, pod, far, csi, cx_rd, ets,
04910     rho_p, rho_s, mean, rmse, absdev, n);
04911
04912 /* Set counters to zero... */
04913 n = ct = cx = cy = cz = 0;
04914 }
04915
04916 /* Close file... */
04917 if (t == ctl->t_stop)
04918     fclose(out);
04919 }
04920
04921 /*****
04922
04923 void write_ens(
04924     const char *filename,
04925     ctl_t * ctl,
04926     atm_t * atm,
04927     double t) {
04928
04929     static FILE *out;
04930
04931     static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
04932         t0, t1, x[NENS][3], xm[3];
04933
04934     static int ip, iq;
04935
04936     static size_t i, n;
04937
04938     /* Set timer... */
04939     SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
04940
04941     /* Init... */
04942     if (t == ctl->t_start) {
04943
04944         /* Check quantities... */
04945         if (ctl->qnt_ens < 0)
04946             ERRMSG("Missing ensemble IDs!");
04947
04948         /* Create new file... */
04949         LOG(1, "Write ensemble data: %s", filename);
04950         if (!out = fopen(filename, "w"))
04951             ERRMSG("Cannot create file!");
04952
04953         /* Write header... */
04954         fprintf(out,
04955             "# $1 = time [s]\n"
04956             "# $2 = altitude [km]\n"
04957             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04958         for (iq = 0; iq < ctl->nq; iq++)
04959             fprintf(out, "# %d = %s (mean) [%s]\n", 5 + iq,
04960                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04961         for (iq = 0; iq < ctl->nq; iq++)
04962             fprintf(out, "# %d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
04963                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04964         fprintf(out, "# %d = number of members\n", 5 + 2 * ctl->nq);
04965     }
04966
04967     /* Set time interval... */
04968     t0 = t - 0.5 * ctl->dt_mod;
04969     t1 = t + 0.5 * ctl->dt_mod;

```

```

04970
04971  /* Init... */
04972  ens = GSL_NAN;
04973  n = 0;
04974
04975  /* Loop over air parcels... */
04976  for (ip = 0; ip < atm->np; ip++) {
04977
04978      /* Check time... */
04979      if (atm->time[ip] < t0 || atm->time[ip] > t1)
04980          continue;
04981
04982      /* Check ensemble id... */
04983      if (atm->q[ctl->qnt_ens][ip] != ens) {
04984
04985          /* Write results... */
04986          if (n > 0) {
04987
04988              /* Get mean position... */
04989              xm[0] = xm[1] = xm[2] = 0;
04990              for (i = 0; i < n; i++) {
04991                  xm[0] += x[i][0] / (double) n;
04992                  xm[1] += x[i][1] / (double) n;
04993                  xm[2] += x[i][2] / (double) n;
04994              }
04995              cart2geo(xm, &dummy, &lon, &lat);
04996              fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
04997                  lat);
04998
04999              /* Get quantity statistics... */
05000              for (iq = 0; iq < ctl->nq; iq++) {
05001                  fprintf(out, " ");
05002                  fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
05003              }
05004              for (iq = 0; iq < ctl->nq; iq++) {
05005                  fprintf(out, " ");
05006                  fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
05007              }
05008              fprintf(out, " %zu\n", n);
05009          }
05010
05011          /* Init new ensemble... */
05012          ens = atm->q[ctl->qnt_ens][ip];
05013          n = 0;
05014      }
05015
05016      /* Save data... */
05017      p[n] = atm->p[ip];
05018      geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
05019      for (iq = 0; iq < ctl->nq; iq++)
05020          q[iq][n] = atm->q[iq][ip];
05021      if ((++n) >= NENS)
05022          ERRMSG("Too many data points!");
05023  }
05024
05025  /* Write results... */
05026  if (n > 0) {
05027
05028      /* Get mean position... */
05029      xm[0] = xm[1] = xm[2] = 0;
05030      for (i = 0; i < n; i++) {
05031          xm[0] += x[i][0] / (double) n;
05032          xm[1] += x[i][1] / (double) n;
05033          xm[2] += x[i][2] / (double) n;
05034      }
05035      cart2geo(xm, &dummy, &lon, &lat);
05036      fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
05037
05038      /* Get quantity statistics... */
05039      for (iq = 0; iq < ctl->nq; iq++) {
05040          fprintf(out, " ");
05041          fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
05042      }
05043      for (iq = 0; iq < ctl->nq; iq++) {
05044          fprintf(out, " ");
05045          fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
05046      }
05047      fprintf(out, " %zu\n", n);
05048  }
05049
05050  /* Close file... */
05051  if (t == ctl->t_stop)
05052      fclose(out);
05053  }
05054
05055  /*****
05056

```

```

05057 void write_grid(
05058     const char *filename,
05059     ctl_t * ctl,
05060     met_t * met0,
05061     met_t * met1,
05062     atm_t * atm,
05063     double t) {
05064
05065     FILE *in, *out;
05066
05067     char line[LEN];
05068
05069     static double mass[GX][GY][GZ], vmr[GX][GY][GZ], vmr_expl, vmr_impl,
05070         z[GZ], dz, lon[GX], dlon, lat[GY], dlat, area[GY], rho_air,
05071         press[GZ], temp, cd, t0, t1, r;
05072
05073     static int ip, ix, *ixs, iy, *iys, iz, *izs, np[GX][GY][GZ], year, mon, day,
05074         hour, min, sec;
05075
05076     /* Set timer... */
05077     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
05078
05079     /* Check dimensions... */
05080     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
05081         ERRMSG("Grid dimensions too large!");
05082
05083     /* Set grid box size... */
05084     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
05085     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
05086     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
05087
05088     /* Set vertical coordinates... */
05089     #pragma omp parallel for default(shared) private(iz)
05090     for (iz = 0; iz < ctl->grid_nz; iz++) {
05091         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
05092         press[iz] = P(z[iz]);
05093     }
05094
05095     /* Set horizontal coordinates... */
05096     for (ix = 0; ix < ctl->grid_nx; ix++)
05097         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
05098     #pragma omp parallel for default(shared) private(iy)
05099     for (iy = 0; iy < ctl->grid_ny; iy++) {
05100         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
05101         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05102             * cos(lat[iy] * M_PI / 180.);
05103     }
05104
05105     /* Set time interval for output... */
05106     t0 = t - 0.5 * ctl->dt_mod;
05107     t1 = t + 0.5 * ctl->dt_mod;
05108
05109     /* Initialize grid... */
05110     #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(3)
05111     for (ix = 0; ix < ctl->grid_nx; ix++)
05112         for (iy = 0; iy < ctl->grid_ny; iy++)
05113             for (iz = 0; iz < ctl->grid_nz; iz++) {
05114                 mass[ix][iy][iz] = 0;
05115                 vmr[ix][iy][iz] = 0;
05116                 np[ix][iy][iz] = 0;
05117             }
05118
05119     /* Allocate... */
05120     ALLOC(ixs, int,
05121         atm->np);
05122     ALLOC(iys, int,
05123         atm->np);
05124     ALLOC(izs, int,
05125         atm->np);
05126
05127     /* Get indices... */
05128     #pragma omp parallel for default(shared) private(ip)
05129     for (ip = 0; ip < atm->np; ip++) {
05130         ix[s[ip]] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
05131         iy[s[ip]] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
05132         iz[s[ip]] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05133         if (atm->time[ip] < t0 || atm->time[ip] > t1
05134             || ix[s[ip]] < 0 || ix[s[ip]] >= ctl->grid_nx
05135             || iy[s[ip]] < 0 || iy[s[ip]] >= ctl->grid_ny
05136             || iz[s[ip]] < 0 || iz[s[ip]] >= ctl->grid_nz)
05137             iz[s[ip]] = -1;
05138     }
05139
05140     /* Average data... */
05141     for (ip = 0; ip < atm->np; ip++)
05142         if (iz[s[ip]] >= 0) {
05143             np[ixs[ip]][iys[ip]][izs[ip]]++;

```

```

05144     if (ctl->qnt_m >= 0)
05145         mass[ixs[ip]][iys[ip]][izs[ip]] += atm->q[ctl->qnt_m][ip];
05146     if (ctl->qnt_vmr >= 0)
05147         vmr[ixs[ip]][iys[ip]][izs[ip]] += atm->q[ctl->qnt_vmr][ip];
05148 }
05149
05150 /* Free... */
05151 free(ixs);
05152 free(iys);
05153 free(izs);
05154
05155 /* Check if gnuplot output is requested... */
05156 if (ctl->grid_gpfile[0] != '-') {
05157
05158     /* Write info... */
05159     LOG(1, "Plot grid data: %s.png", filename);
05160
05161     /* Create gnuplot pipe... */
05162     if (!(out = popen("gnuplot", "w")))
05163         ERRMSG("Cannot create pipe to gnuplot!");
05164
05165     /* Set plot filename... */
05166     fprintf(out, "set out \"%s.png\\n\", filename);
05167
05168     /* Set time string... */
05169     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05170     fprintf(out, "timestr=\"%d-%02d-%02d %02d:%02d UTC\\n\",
05171             year, mon, day, hour, min);
05172
05173     /* Dump gnuplot file to pipe... */
05174     if (!(in = fopen(ctl->grid_gpfile, "r")))
05175         ERRMSG("Cannot open file!");
05176     while (fgets(line, LEN, in))
05177         fprintf(out, "%s", line);
05178     fclose(in);
05179 }
05180
05181 else {
05182
05183     /* Write info... */
05184     LOG(1, "Write grid data: %s", filename);
05185
05186     /* Create file... */
05187     if (!(out = fopen(filename, "w")))
05188         ERRMSG("Cannot create file!");
05189 }
05190
05191 /* Write header... */
05192 fprintf(out,
05193         "# $1 = time [s]\\n"
05194         "# $2 = altitude [km]\\n"
05195         "# $3 = longitude [deg]\\n"
05196         "# $4 = latitude [deg]\\n"
05197         "# $5 = surface area [km^2]\\n"
05198         "# $6 = layer width [km]\\n"
05199         "# $7 = number of particles [l]\\n"
05200         "# $8 = column density (implicit) [kg/m^2]\\n"
05201         "# $9 = volume mixing ratio (implicit) [ppv]\\n"
05202         "# $10 = volume mixing ratio (explicit) [ppv]\\n\\n");
05203
05204 /* Write data... */
05205 for (ix = 0; ix < ctl->grid_nx; ix++) {
05206     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
05207         fprintf(out, "\\n");
05208     for (iy = 0; iy < ctl->grid_ny; iy++) {
05209         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
05210             fprintf(out, "\\n");
05211         for (iz = 0; iz < ctl->grid_nz; iz++) {
05212             if (!ctl->grid_sparse || mass[ix][iy][iz] > 0 || vmr[ix][iy][iz] > 0) {
05213
05214                 /* Calculate column density... */
05215                 if (ctl->qnt_m >= 0)
05216                     cd = mass[ix][iy][iz] / (1e6 * area[iy]);
05217                 else
05218                     cd = GSL_NAN;
05219
05220                 /* Calculate volume mixing ratio (implicit)... */
05221                 if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05222                     vmr_impl = 0;
05223                     if (mass[ix][iy][iz] > 0) {
05224
05225                         /* Get temperature... */
05226                         INTPOL_INIT;
05227                         intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05228                                           lon[ix], lat[iy], &temp, ci, cw, 1);
05229
05230                         /* Calculate density of air... */

```

```

05231         rho_air = 100. * press[iz] / (RA * temp);
05232
05233         /* Calculate volume mixing ratio... */
05234         vmr_impl = MA / ctl->molmass * mass[ix][iy][iz]
05235             / (rho_air * 1e6 * area[iy] * 1e3 * dz);
05236     }
05237 } else
05238     vmr_impl = GSL_NAN;
05239
05240     /* Calculate volume mixing ratio (explicit)... */
05241     if (ctl->qnt_vmr >= 0 && np[ix][iy][iz] > 0)
05242         vmr_expl = vmr[ix][iy][iz] / np[ix][iy][iz];
05243     else
05244         vmr_expl = GSL_NAN;
05245
05246     /* Write output... */
05247     fprintf(out, "%.2f %g %g %g %g %d %g %g %g\n", t, z[iz],
05248         lon[ix], lat[iy], area[iy], dz, np[ix][iy][iz], cd,
05249         vmr_impl, vmr_expl);
05250 }
05251 }
05252 }
05253
05254 /* Close file... */
05255 fclose(out);
05256 }
05257
05258 /*****
05259 void write_prof(
05260     const char *filename,
05261     ctl_t * ctl,
05262     met_t * met0,
05263     met_t * met1,
05264     atm_t * atm,
05265     double t) {
05266
05267     static FILE *in, *out;
05268
05269     static char line[LEN];
05270
05271     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rt_old,
05272         rz, rlon, rlat, robs, t0, t1, area[GY], dz, dlon, dlat,
05273         lon[GX], lat[GY], z[GZ], press[GZ], temp, rho_air, vmr, h2o, o3;
05274
05275     static int obscount[GX][GY], ip, ix, iy, iz, okay;
05276
05277     /* Set timer... */
05278     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
05279
05280     /* Init... */
05281     if (t == ctl->t_start) {
05282
05283         /* Check quantity index for mass... */
05284         if (ctl->qnt_m < 0)
05285             ERRMSG("Need quantity mass!");
05286
05287         /* Check dimensions... */
05288         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
05289             ERRMSG("Grid dimensions too large!");
05290
05291         /* Check molar mass... */
05292         if (ctl->molmass <= 0)
05293             ERRMSG("Specify molar mass!");
05294
05295         /* Open observation data file... */
05296         LOG(1, "Read profile observation data: %s", ctl->prof_obsfile);
05297         if (!(in = fopen(ctl->prof_obsfile, "r")))
05298             ERRMSG("Cannot open file!");
05299
05300         /* Initialize time for file input... */
05301         rt_old = -1e99;
05302
05303         /* Create new output file... */
05304         LOG(1, "Write profile data: %s", filename);
05305         if (!(out = fopen(filename, "w")))
05306             ERRMSG("Cannot create file!");
05307
05308         /* Write header... */
05309         fprintf(out,
05310             "# $1 = time [s]\n"
05311             "# $2 = altitude [km]\n"
05312             "# $3 = longitude [deg]\n"
05313             "# $4 = latitude [deg]\n"
05314             "# $5 = pressure [hPa]\n"
05315             "# $6 = temperature [K]\n"
05316             "# $7 = volume mixing ratio [ppv]\n"

```



```

05318         "# $8 = H2O volume mixing ratio [ppv]\n"
05319         "# $9 = O3 volume mixing ratio [ppv]\n"
05320         "# $10 = observed BT index [K]\n"
05321         "# $11 = number of observations\n");
05322
05323     /* Set grid box size... */
05324     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
05325     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
05326     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
05327
05328     /* Set vertical coordinates... */
05329     for (iz = 0; iz < ctl->prof_nz; iz++) {
05330         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
05331         press[iz] = P(z[iz]);
05332     }
05333
05334     /* Set horizontal coordinates... */
05335     for (ix = 0; ix < ctl->prof_nx; ix++)
05336         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
05337     for (iy = 0; iy < ctl->prof_ny; iy++) {
05338         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
05339         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05340             * cos(lat[iy] * M_PI / 180.);
05341     }
05342 }
05343
05344 /* Set time interval... */
05345 t0 = t - 0.5 * ctl->dt_mod;
05346 t1 = t + 0.5 * ctl->dt_mod;
05347
05348 /* Initialize... */
05349 #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(2)
05350 for (ix = 0; ix < ctl->prof_nx; ix++)
05351     for (iy = 0; iy < ctl->prof_ny; iy++) {
05352         obsmean[ix][iy] = 0;
05353         obscount[ix][iy] = 0;
05354         for (iz = 0; iz < ctl->prof_nz; iz++)
05355             mass[ix][iy][iz] = 0;
05356     }
05357
05358 /* Read observation data... */
05359 while (fgets(line, LEN, in)) {
05360
05361     /* Read data... */
05362     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
05363         5)
05364         continue;
05365
05366     /* Check time... */
05367     if (rt < t0)
05368         continue;
05369     if (rt > t1)
05370         break;
05371     if (rt < rt_old)
05372         ERRMSG("Time must be ascending!");
05373     rt_old = rt;
05374
05375     /* Check observation data... */
05376     if (!isfinite(robs))
05377         continue;
05378
05379     /* Calculate indices... */
05380     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
05381     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
05382
05383     /* Check indices... */
05384     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
05385         continue;
05386
05387     /* Get mean observation index... */
05388     obsmean[ix][iy] += robs;
05389
05390     /* Count observations... */
05391     obscount[ix][iy]++;
05392 }
05393
05394 /* Analyze model data... */
05395 for (ip = 0; ip < atm->np; ip++) {
05396
05397     /* Check time... */
05398     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05399         continue;
05400
05401     /* Get indices... */
05402     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
05403     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
05404     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);

```

```

05405
05406     /* Check indices... */
05407     if (ix < 0 || ix >= ctl->prof_nx ||
05408         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
05409         continue;
05410
05411     /* Get total mass in grid cell... */
05412     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
05413 }
05414
05415 /* Extract profiles... */
05416 for (ix = 0; ix < ctl->prof_nx; ix++)
05417     for (iy = 0; iy < ctl->prof_ny; iy++)
05418         if (obscount[ix][iy] >= 1) {
05419
05420             /* Check profile... */
05421             okay = 0;
05422             for (iz = 0; iz < ctl->prof_nz; iz++)
05423                 if (mass[ix][iy][iz] > 0) {
05424                     okay = 1;
05425                     break;
05426                 }
05427             if (!okay)
05428                 continue;
05429
05430             /* Write output... */
05431             fprintf(out, "\n");
05432
05433             /* Loop over altitudes... */
05434             for (iz = 0; iz < ctl->prof_nz; iz++) {
05435
05436                 /* Get pressure and temperature... */
05437                 INTPOL_INIT;
05438                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05439                                     lon[ix], lat[iy], &temp, ci, cw, 1);
05440                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
05441                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
05442                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
05443                                     lon[ix], lat[iy], &o3, ci, cw, 0);
05444
05445                 /* Calculate volume mixing ratio... */
05446                 rho_air = 100. * press[iz] / (RA * temp);
05447                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
05448                     / (rho_air * area[iy] * dz * 1e9);
05449
05450                 /* Write output... */
05451                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
05452                         t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
05453                         obsmean[ix][iy] / obscount[ix][iy], obscount[ix][iy]);
05454             }
05455         }
05456
05457 /* Close files... */
05458 if (t == ctl->t_stop) {
05459     fclose(in);
05460     fclose(out);
05461 }
05462 }
05463
05464 /*****
05465
05466 void write_sample(
05467     const char *filename,
05468     ctl_t * ctl,
05469     met_t * met0,
05470     met_t * met1,
05471     atm_t * atm,
05472     double t) {
05473
05474     static FILE *in, *out;
05475
05476     static char line[LEN];
05477
05478     static double area, dlat, rmax2, t0, t1, rt, rt_old, rz, rlon, rlat, robs;
05479
05480     /* Set timer... */
05481     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
05482
05483     /* Init... */
05484     if (t == ctl->t_start) {
05485
05486         /* Open observation data file... */
05487         LOG(1, "Read sample observation data: %s", ctl->sample_obsfile);
05488         if (!(in = fopen(ctl->sample_obsfile, "r")))
05489             ERRMSG("Cannot open file!");
05490
05491         /* Initialize time for file input... */

```

```

05492     rt_old = -1e99;
05493
05494     /* Create new file... */
05495     LOG(1, "Write sample data: %s", filename);
05496     if (! (out = fopen(filename, "w")))
05497         ERRMSG("Cannot create file!");
05498
05499     /* Write header... */
05500     fprintf(out,
05501         "# $1 = time [s]\n"
05502         "# $2 = altitude [km]\n"
05503         "# $3 = longitude [deg]\n"
05504         "# $4 = latitude [deg]\n"
05505         "# $5 = surface area [km^2]\n"
05506         "# $6 = layer width [km]\n"
05507         "# $7 = number of particles [l]\n"
05508         "# $8 = column density [kg/m^2]\n"
05509         "# $9 = volume mixing ratio [ppv]\n"
05510         "# $10 = observed BT index [K]\n\n");
05511
05512     /* Set latitude range, squared radius, and area... */
05513     dlat = DY2DEG(ctl->sample_dx);
05514     rmax2 = SQR(ctl->sample_dx);
05515     area = M_PI * rmax2;
05516 }
05517
05518 /* Set time interval for output... */
05519 t0 = t - 0.5 * ctl->dt_mod;
05520 t1 = t + 0.5 * ctl->dt_mod;
05521
05522 /* Read observation data... */
05523 while (fgets(line, LEN, in)) {
05524
05525     /* Read data... */
05526     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
05527         5)
05528         continue;
05529
05530     /* Check time... */
05531     if (rt < t0)
05532         continue;
05533     if (rt < rt_old)
05534         ERRMSG("Time must be ascending!");
05535     rt_old = rt;
05536
05537     /* Calculate Cartesian coordinates... */
05538     double x0[3];
05539     geo2cart(0, rln, rln, x0);
05540
05541     /* Set pressure range... */
05542     double rp = P(rz);
05543     double ptop = P(rz + ctl->sample_dz);
05544     double pbot = P(rz - ctl->sample_dz);
05545
05546     /* Init... */
05547     double mass = 0;
05548     int np = 0;
05549
05550     /* Loop over air parcels... */
05551 #pragma omp parallel for default(shared) reduction(+:mass,np)
05552     for (int ip = 0; ip < atm->np; ip++) {
05553
05554         /* Check time... */
05555         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05556             continue;
05557
05558         /* Check latitude... */
05559         if (fabs(rln - atm->lat[ip]) > dlat)
05560             continue;
05561
05562         /* Check horizontal distance... */
05563         double x1[3];
05564         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05565         if (DIST2(x0, x1) > rmax2)
05566             continue;
05567
05568         /* Check pressure... */
05569         if (ctl->sample_dz > 0)
05570             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
05571                 continue;
05572
05573         /* Add mass... */
05574         if (ctl->qnt_m >= 0)
05575             mass += atm->q[ctl->qnt_m][ip];
05576         np++;
05577     }
05578

```

```

05579      /* Calculate column density... */
05580      double cd = mass / (1e6 * area);
05581
05582      /* Calculate volume mixing ratio... */
05583      double vmr = 0;
05584      if (ctl->molmass > 0 && ctl->sample_dz > 0) {
05585          if (mass > 0) {
05586
05587              /* Get temperature... */
05588              double temp;
05589              INTPOL_INIT;
05590              intpol_met_time_3d(met0, met0->t, met1, met1->t, rt, rp,
05591                              rlon, rlat, &temp, ci, cw, 1);
05592
05593              /* Calculate density of air... */
05594              double rho_air = 100. * rp / (RA * temp);
05595
05596              /* Calculate volume mixing ratio... */
05597              vmr = MA / ctl->molmass * mass
05598                  / (rho_air * 1e6 * area * 1e3 * ctl->sample_dz);
05599          }
05600      } else
05601          vmr = GSL_NAN;
05602
05603      /* Write output... */
05604      fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt, rz, rlon, rlat,
05605              area, ctl->sample_dz, np, cd, vmr, robs);
05606
05607      /* Check time... */
05608      if (rt >= t1)
05609          break;
05610  }
05611
05612  /* Close files... */
05613  if (t == ctl->t_stop) {
05614      fclose(in);
05615      fclose(out);
05616  }
05617 }
05618
05619 /*****
05620
05621 void write_station(
05622     const char *filename,
05623     ctl_t * ctl,
05624     atm_t * atm,
05625     double t) {
05626
05627     static FILE *out;
05628
05629     static double rmax2, t0, t1, x0[3], x1[3];
05630
05631     /* Set timer... */
05632     SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
05633
05634     /* Init... */
05635     if (t == ctl->t_start) {
05636
05637         /* Write info... */
05638         LOG(1, "Write station data: %s", filename);
05639
05640         /* Create new file... */
05641         if (!(out = fopen(filename, "w")))
05642             ERRMSG("Cannot create file!");
05643
05644         /* Write header... */
05645         fprintf(out,
05646             "# $1 = time [s]\n"
05647             "# $2 = altitude [km]\n"
05648             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05649         for (int iq = 0; iq < ctl->nq; iq++)
05650             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
05651                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05652         fprintf(out, "\n");
05653
05654         /* Set geolocation and search radius... */
05655         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
05656         rmax2 = SQR(ctl->stat_r);
05657     }
05658
05659     /* Set time interval for output... */
05660     t0 = t - 0.5 * ctl->dt_mod;
05661     t1 = t + 0.5 * ctl->dt_mod;
05662
05663     /* Loop over air parcels... */
05664     for (int ip = 0; ip < atm->np; ip++) {
05665

```

```

05666     /* Check time... */
05667     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05668         continue;
05669
05670     /* Check time range for station output... */
05671     if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
05672         continue;
05673
05674     /* Check station flag... */
05675     if (ctl->qnt_stat >= 0)
05676         if (atm->q[ctl->qnt_stat][ip])
05677             continue;
05678
05679     /* Get Cartesian coordinates... */
05680     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05681
05682     /* Check horizontal distance... */
05683     if (DIST2(x0, x1) > rmax2)
05684         continue;
05685
05686     /* Set station flag... */
05687     if (ctl->qnt_stat >= 0)
05688         atm->q[ctl->qnt_stat][ip] = 1;
05689
05690     /* Write data... */
05691     fprintf(out, "%.2f %g %g %g",
05692            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
05693     for (int iq = 0; iq < ctl->nq; iq++) {
05694         fprintf(out, " ");
05695         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05696     }
05697     fprintf(out, "\n");
05698 }
05699
05700 /* Close file... */
05701 if (t == ctl->t_stop)
05702     fclose(out);
05703 }

```

5.21 libtrac.h File Reference

```

#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_sort.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_statistics.h>
#include <math.h>
#include <netcdf.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

```

Data Structures

- struct [ctl_t](#)
Control parameters.
- struct [atm_t](#)
Atmospheric data.
- struct [cache_t](#)
Cache data.
- struct [met_t](#)
Meteorological 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/s²].
- #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 m²/(K s²)].
- #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 EP 140
Maximum number of pressure levels for meteorological data.
- #define EX 1201
Maximum number of longitudes for meteorological data.
- #define EY 601
Maximum number of latitudes for meteorological data.
- #define GX 720
Maximum number of longitudes for gridded data.
- #define GY 360
Maximum number of latitudes for gridded data.
- #define GZ 100
Maximum number of altitudes for gridded data.
- #define NENS 2000
Maximum number of data points for ensemble analysis.
- #define NTHREADS 512

- Maximum number of OpenMP threads.*

 - #define **ALLOC**(ptr, type, n)

Allocate and clear memory.
- #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 **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) (0.01 * pow(10., -2663.5 / (t) + 12.537))
Compute saturation pressure over ice (Marti and Mauersberger, 1993).
- #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 SET_ATM(qnt, val)
Set atmospheric quantity value.
- #define SET_QNT(qnt, name, 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 TDEW(p, h2o)
Calculate dew point temperature (WMO, 2018).
- #define TICE(p, h2o) (-2663.5 / (log10(100. * PW((p), (h2o)))) - 12.537))
Calculate frost point temperature (Marti and Mauersberger, 1993).
- #define THETA(p, t) ((t) * pow(1000. / (p), 0.286))
Compute potential temperature.
- #define THETAVRT(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 `cart2geo` (double *x, double *z, double *lon, double *lat)
Convert Cartesian coordinates to geolocation.
- int `check_finite` (const double x)
Check if x is finite.
- double `clim_hno3` (double t, double lat, double p)
Climatology of HNO₃ volume mixing ratios.
- double `clim_oh` (double t, double lat, double p)
Climatology of OH number concentrations.
- double `clim_tropo` (double t, double lat)
Climatology of tropopause pressure.
- void `day2doy` (int year, int mon, int day, int *doy)
Get day of year from date.
- void `doy2day` (int year, int doy, int *mon, int *day)
Get date from day of year.
- void `geo2cart` (double z, double lon, double lat, double *x)
Convert geolocation to Cartesian coordinates.
- void `get_met` (ctl_t *ctl, double t, met_t **met0, met_t **met1)
Get meteorological data for given time step.
- void `get_met_help` (double t, int direct, char *metbase, double dt_met, char *filename)
Get meteorological 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 meteorological data.
- void `intpol_met_space_2d` (met_t *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteorological data.
- void `intpol_met_time_3d` (met_t *met0, float array0[EX][EY][EP], met_t *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void `intpol_met_time_2d` (met_t *met0, float array0[EX][EY], met_t *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void `jsec2time` (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)
Convert seconds to date.
- 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.*

 - int `read_atm` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
- Read atmospheric data.*

 - void `read_ctl` (const char *filename, int argc, char *argv[], `ctl_t` *ctl)
- Read control parameters.*

 - int `read_met` (`ctl_t` *ctl, char *filename, `met_t` *met)
- Read meteorological data file.*

 - void `read_met_cape` (`met_t` *met)
- Calculate convective available potential energy.*

 - void `read_met_cloud` (`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 meteorological 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 meteorological data.*

 - int `read_met_help_3d` (int ncid, char *varname, char *varname2, `met_t` *met, float dest[EX][EY][EP], float scl, int init)
- Read and convert 3D variable from meteorological data file.*

 - int `read_met_help_2d` (int ncid, char *varname, char *varname2, `met_t` *met, float dest[EX][EY], float scl, int init)
- Read and convert 2D variable from meteorological data file.*

 - void `read_met_levels` (int ncid, `ctl_t` *ctl, `met_t` *met)
- Read meteorological data on vertical levels.*

 - void `read_met_ml2pl` (`ctl_t` *ctl, `met_t` *met, float var[EX][EY][EP])
- Convert meteorological data from model levels to pressure levels.*

 - void `read_met_pbl` (`met_t` *met)
- Calculate pressure of the boundary layer.*

 - void `read_met_periodic` (`met_t` *met)
- Create meteorological data with periodic boundary conditions.*

 - void `read_met_pv` (`met_t` *met)
- Calculate potential vorticity.*

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

 - void `read_met_surface` (int ncid, `met_t` *met)
- Read surface data.*

 - void `read_met_tropo` (`ctl_t` *ctl, `met_t` *met)
- Calculate tropopause 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 r_p, double rho_p)
- 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.*

 - 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](#) (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_csi](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)

Write CSI data.

- void [write_ens](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)

Write ensemble data.

- void [write_grid](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)

Write gridded data.

- void [write_prof](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)

Write profile data.

- void [write_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 declarations.

Definition in file [libtrac.h](#).

5.21.2 Macro Definition Documentation

5.21.2.1 CPD `#define CPD 1003.5`

Specific heat of dry air at constant pressure [J/(kg K)].

Definition at line 74 of file [libtrac.h](#).

5.21.2.2 EPS `#define EPS (MH2O / MA)`

Ratio of the specific gas constant of dry air and water vapor [1].

Definition at line 77 of file [libtrac.h](#).

5.21.2.3 G0 `#define G0 9.80665`

Standard gravity [m/s²].

Definition at line 80 of file [libtrac.h](#).

5.21.2.4 H0 `#define H0 7.0`

Scale height [km].

Definition at line 83 of file [libtrac.h](#).

5.21.2.5 LV `#define LV 2501000.`

Latent heat of vaporization of water [J/kg].

Definition at line 86 of file [libtrac.h](#).

5.21.2.6 KB `#define KB 1.3806504e-23`

Boltzmann constant [kg m²/(K s²)].

Definition at line 89 of file [libtrac.h](#).

5.21.2.7 MA `#define MA 28.9644`

Molar mass of dry air [g/mol].

Definition at line 92 of file [libtrac.h](#).

5.21.2.8 MH2O `#define MH2O 18.01528`

Molar mass of water vapor [g/mol].

Definition at line 95 of file [libtrac.h](#).

5.21.2.9 MO3 `#define MO3 48.00`

Molar mass of ozone [g/mol].

Definition at line 98 of file [libtrac.h](#).

5.21.2.10 P0 `#define P0 1013.25`

Standard pressure [hPa].

Definition at line 101 of file [libtrac.h](#).

5.21.2.11 RA `#define RA (1e3 * RI / MA)`

Specific gas constant of dry air [J/(kg K)].

Definition at line 104 of file [libtrac.h](#).

5.21.2.12 RE `#define RE 6367.421`

Mean radius of Earth [km].

Definition at line 107 of file [libtrac.h](#).

5.21.2.13 RI `#define RI 8.3144598`

Ideal gas constant [J/(mol K)].

Definition at line 110 of file [libtrac.h](#).

5.21.2.14 T0 `#define T0 273.15`

Standard temperature [K].

Definition at line 113 of file [libtrac.h](#).

5.21.2.15 LEN `#define LEN 5000`

Maximum length of ASCII data lines.

Definition at line [121](#) of file [libtrac.h](#).

5.21.2.16 NP `#define NP 10000000`

Maximum number of atmospheric data points.

Definition at line [126](#) of file [libtrac.h](#).

5.21.2.17 NQ `#define NQ 15`

Maximum number of quantities per data point.

Definition at line [131](#) of file [libtrac.h](#).

5.21.2.18 EP `#define EP 140`

Maximum number of pressure levels for meteorological data.

Definition at line [136](#) of file [libtrac.h](#).

5.21.2.19 EX `#define EX 1201`

Maximum number of longitudes for meteorological data.

Definition at line [141](#) of file [libtrac.h](#).

5.21.2.20 EY `#define EY 601`

Maximum number of latitudes for meteorological data.

Definition at line [146](#) of file [libtrac.h](#).

5.21.2.21 GX `#define GX 720`

Maximum number of longitudes for gridded data.

Definition at line 151 of file [libtrac.h](#).

5.21.2.22 GY `#define GY 360`

Maximum number of latitudes for gridded data.

Definition at line 156 of file [libtrac.h](#).

5.21.2.23 GZ `#define GZ 100`

Maximum number of altitudes for gridded data.

Definition at line 161 of file [libtrac.h](#).

5.21.2.24 NENS `#define NENS 2000`

Maximum number of data points for ensemble analysis.

Definition at line 166 of file [libtrac.h](#).

5.21.2.25 NTHREADS `#define NTHREADS 512`

Maximum number of OpenMP threads.

Definition at line 171 of file [libtrac.h](#).

5.21.2.26 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 186 of file [libtrac.h](#).

5.21.2.27 DEG2DX `#define DEG2DX(
 dlon,
 lat) ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))`

Convert degrees to zonal distance.

Definition at line 192 of file [libtrac.h](#).

5.21.2.28 DEG2DY `#define DEG2DY(
 dlat) ((dlat) * M_PI * RE / 180.)`

Convert degrees to meridional distance.

Definition at line 196 of file [libtrac.h](#).

5.21.2.29 DP2DZ `#define DP2DZ(
 dp,
 p) (- (dp) * H0 / (p))`

Convert pressure change to vertical distance.

Definition at line 200 of file [libtrac.h](#).

5.21.2.30 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 204 of file [libtrac.h](#).

5.21.2.31 DY2DEG `#define DY2DEG(
 dy) ((dy) * 180. / (M_PI * RE))`

Convert meridional distance to degrees.

Definition at line 209 of file [libtrac.h](#).

5.21.2.32 DZ2DP `#define DZ2DP(
 dz,
 p) (-(dz) * (p) / H0)`

Convert vertical distance to pressure change.

Definition at line 213 of file [libtrac.h](#).

5.21.2.33 DIST `#define DIST(
 a,
 b) sqrt(DIST2(a, b))`

Compute Cartesian distance between two vectors.

Definition at line 217 of file [libtrac.h](#).

5.21.2.34 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 221 of file [libtrac.h](#).

5.21.2.35 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 225 of file [libtrac.h](#).

5.21.2.36 FMOD `#define FMOD(
 x,
 y) ((x) - (int) ((x) / (y)) * (y))`

Compute floating point modulo.

Definition at line 229 of file [libtrac.h](#).

5.21.2.37 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 [233](#) of file [libtrac.h](#).

5.21.2.38 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 [239](#) of file [libtrac.h](#).

5.21.2.39 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 [245](#) of file [libtrac.h](#).

5.21.2.40 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 [249](#) of file [libtrac.h](#).

5.21.2.41 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 255 of file [libtrac.h](#).

5.21.2.42 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->c1, lon, lat, &c1, 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 262 of file [libtrac.h](#).

5.21.2.43 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->c1, met1, met1->c1, time, lon, lat, &c1, 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 294 of file [libtrac.h](#).

```

5.21.2.44 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 326 of file [libtrac.h](#).

```

5.21.2.45 LIN #define LIN(
    x0,
    y0,
    x1,
    y1,
    x ) ((y0) + ((y1) - (y0)) / ((x1) - (x0)) * ((x) - (x0)))

```

Compute linear interpolation.

Definition at line 331 of file [libtrac.h](#).

5.21.2.46 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 335 of file [libtrac.h](#).

5.21.2.47 NN `#define NN(
 x0,
 y0,
 x1,
 y1,
 x) (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))`

Compute nearest neighbor interpolation.

Definition at line 342 of file [libtrac.h](#).

5.21.2.48 NORM `#define NORM(
 a) sqrt(DOTP(a, a))`

Compute norm of a vector.

Definition at line 346 of file [libtrac.h](#).

5.21.2.49 P `#define P(
 z) (P0 * exp(-(z) / H0))`

Convert altitude to pressure.

Definition at line 350 of file [libtrac.h](#).

5.21.2.50 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 354 of file [libtrac.h](#).

5.21.2.51 PSICE `#define PSICE(
 t) (0.01 * pow(10., -2663.5 / (t) + 12.537))`

Compute saturation pressure over ice (Marti and Mauersberger, 1993).

Definition at line 358 of file [libtrac.h](#).

5.21.2.52 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 362 of file [libtrac.h](#).

5.21.2.53 RH `#define RH(
 p,
 t,
 h2o) (PW(p, h2o) / PSAT(t) * 100.)`

Compute relative humidity over water.

Definition at line 367 of file [libtrac.h](#).

5.21.2.54 RHICE `#define RHICE(
 p,
 t,
 h2o) (PW(p, h2o) / PSICE(t) * 100.)`

Compute relative humidity over ice.

Definition at line 371 of file [libtrac.h](#).

5.21.2.55 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 375 of file [libtrac.h](#).

5.21.2.56 SET_QNT `#define SET_QNT(
 qnt,
 name,
 unit)`

Value:

```
if (strcasecmp(ctl->qnt_name[iq], name) == 0) { \
    ctl->qnt = iq; \
    sprintf(ctl->qnt_unit[iq], unit); \
} else
```

Set atmospheric quantity index.

Definition at line 380 of file [libtrac.h](#).

5.21.2.57 SH `#define SH(
 h2o) (EPS * GSL_MAX((h2o), 0.1e-6))`

Compute specific humidity from water vapor volume mixing ratio.

Definition at line 387 of file [libtrac.h](#).

5.21.2.58 SQR `#define SQR(
 x) ((x)*(x))`

Compute square.

Definition at line 391 of file [libtrac.h](#).

5.21.2.59 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 395 of file [libtrac.h](#).

5.21.2.60 TICE `#define TICE(
 P,
 h2o) (-2663.5 / (log10(100. * PW((p), (h2o))) - 12.537))`

Calculate frost point temperature (Marti and Mauersberger, 1993).

Definition at line 400 of file [libtrac.h](#).

5.21.2.61 THETA `#define THETA(
 p,
 t) ((t) * pow(1000. / (p), 0.286))`

Compute potential temperature.

Definition at line 404 of file [libtrac.h](#).

5.21.2.62 THETA_VIRT `#define THETA_VIRT(
 p,
 t,
 h2o) (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))`

Compute virtual potential temperature.

Definition at line 408 of file [libtrac.h](#).

5.21.2.63 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 412 of file [libtrac.h](#).

5.21.2.64 TVIRT `#define TVIRT(
 t,
 h2o) ((t) * (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))`

Compute virtual temperature.

Definition at line 419 of file [libtrac.h](#).

5.21.2.65 Z `#define Z(
 p) (H0 * log(P0 / (p)))`

Convert pressure to altitude.

Definition at line 423 of file [libtrac.h](#).

5.21.2.66 ZDIFF `#define ZDIFF(
 lnp0,
 t0,
 h2o0,
 lnp1,
 t1,
 h2o1)`

Value:

```
(RI / MA / GO * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1)))  

 * ((lnp0) - (lnp1))) \
```

Calculate geopotential height difference.

Definition at line 427 of file [libtrac.h](#).

5.21.2.67 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 432 of file [libtrac.h](#).

5.21.2.68 LOGLEV `#define LOGLEV 2`

Level of log messages (0=none, 1=basic, 2=detailed, 3=debug).

Definition at line 443 of file [libtrac.h](#).

5.21.2.69 LOG `#define LOG(
 level,
 ...)`

Value:

```
{  

 if(level >= 2) \  

     printf(" ");  

 if(level <= LOGLEV) {  

     printf(__VA_ARGS__);  

     printf("\n");  

 }  

 } \
```

Print log message.

Definition at line 447 of file [libtrac.h](#).

5.21.2.70 WARN `#define WARN(
 ...)`

Value:

```
{  
    printf("\nWarning (%s, %s, l%d): ", __FILE__, __func__, __LINE__); \  
    LOG(0, __VA_ARGS__); \  
}
```

Print warning message.

Definition at line [457](#) of file [libtrac.h](#).

5.21.2.71 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 [463](#) of file [libtrac.h](#).

5.21.2.72 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 [470](#) of file [libtrac.h](#).

5.21.2.73 NTIMER `#define NTIMER 100`

Maximum number of timers.

Definition at line [479](#) of file [libtrac.h](#).

5.21.2.74 PRINT_TIMERS `#define PRINT_TIMERS timer("END", "END", 1);`

Print timers.

Definition at line [482](#) of file [libtrac.h](#).

5.21.2.75 SELECT_TIMER `#define SELECT_TIMER(
 id,
 group,
 color)`

Value:

```
{  
    NVTX_POP; \br/>    NVTX_PUSH(id, color); \br/>    timer(id, group, 0); \br/>}
```

Select timer.

Definition at line [486](#) of file [libtrac.h](#).

5.21.2.76 START_TIMERS `#define START_TIMERS NVTX_PUSH("START", NVTX_CPU);`

Start timers.

Definition at line [493](#) of file [libtrac.h](#).

5.21.2.77 STOP_TIMERS `#define STOP_TIMERS NVTX_POP;`

Stop timers.

Definition at line [497](#) of file [libtrac.h](#).

5.21.2.78 NVTX_PUSH `#define NVTX_PUSH(
 range_title,
 range_color) {}`

Definition at line [544](#) of file [libtrac.h](#).

5.21.2.79 NVTX_POP `#define NVTX_POP {}`

Definition at line [545](#) of file [libtrac.h](#).

5.21.3 Function Documentation

5.21.3.1 cart2geo() void cart2geo (

```

    double * x,
    double * z,
    double * lon,
    double * lat )

```

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file [libtrac.c](#).

```

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

```

5.21.3.2 check_finite() int check_finite (

```

    const double x )

```

Check if x is finite.

5.21.3.3 clim_hno3() double clim_hno3 (

```

    double t,
    double lat,
    double p )

```

Climatology of HNO3 volume mixing ratios.

Definition at line 295 of file [libtrac.c](#).

```

00298     {
00299
00300     /* Get seconds since begin of year... */
00301     double sec = FMOD(t, 365.25 * 86400.);
00302     while (sec < 0)
00303         sec += 365.25 * 86400.;
00304
00305     /* Check pressure... */
00306     if (p < clim_hno3_ps[0])
00307         p = clim_hno3_ps[0];
00308     else if (p > clim_hno3_ps[9])
00309         p = clim_hno3_ps[9];
00310
00311     /* Check latitude... */
00312     if (lat < clim_hno3_lats[0])
00313         lat = clim_hno3_lats[0];
00314     else if (lat > clim_hno3_lats[17])
00315         lat = clim_hno3_lats[17];
00316
00317     /* Get indices... */
00318     int isec = locate_irr(clim_hno3_secs, 12, sec);
00319     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00320     int ip = locate_irr(clim_hno3_ps, 10, p);
00321
00322     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00323     double aux00 = LIN(clim_hno3_ps[ip],
00324                        clim_hno3_var[isec][ilat][ip],
00325                        clim_hno3_ps[ip + 1],
00326                        clim_hno3_var[isec][ilat][ip + 1], p);
00327     double aux01 = LIN(clim_hno3_ps[ip],
00328                        clim_hno3_var[isec][ilat + 1][ip],
00329                        clim_hno3_ps[ip + 1],
00330                        clim_hno3_var[isec][ilat + 1][ip + 1], p);
00331     double aux10 = LIN(clim_hno3_ps[ip],
00332                        clim_hno3_var[isec + 1][ilat][ip],
00333                        clim_hno3_ps[ip + 1],
00334                        clim_hno3_var[isec + 1][ilat][ip + 1], p);

```

```

00335 double aux11 = LIN(clim_hno3_ps[ip],
00336                   clim_hno3_var[isec + 1][ilat + 1][ip],
00337                   clim_hno3_ps[ip + 1],
00338                   clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00339 aux00 = LIN(clim_hno3_lats[ilat], aux00,
00340            clim_hno3_lats[ilat + 1], aux01, lat);
00341 aux11 = LIN(clim_hno3_lats[ilat], aux10,
00342            clim_hno3_lats[ilat + 1], aux11, lat);
00343 aux00 = LIN(clim_hno3_secs[isec], aux00,
00344            clim_hno3_secs[isec + 1], aux11, sec);
00345
00346 /* Convert from ppb to ppv... */
00347 return GSL_MAX(1e-9 * aux00, 0.0);
00348 }

```

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

Climatology of OH number concentrations.

Definition at line 1331 of file libtrac.c.

```

01334 {
01335
01336 /* Get seconds since begin of year... */
01337 double sec = FMOD(t, 365.25 * 86400.);
01338 while (sec < 0)
01339     sec += 365.25 * 86400.;
01340
01341 /* Check pressure... */
01342 if (p < clim_oh_ps[0])
01343     p = clim_oh_ps[0];
01344 else if (p > clim_oh_ps[33])
01345     p = clim_oh_ps[33];
01346
01347 /* Check latitude... */
01348 if (lat < clim_oh_lats[0])
01349     lat = clim_oh_lats[0];
01350 else if (lat > clim_oh_lats[17])
01351     lat = clim_oh_lats[17];
01352
01353 /* Get indices... */
01354 int isec = locate_irr(clim_oh_secs, 12, sec);
01355 int ilat = locate_reg(clim_oh_lats, 18, lat);
01356 int ip = locate_irr(clim_oh_ps, 34, p);
01357
01358 /* Interpolate OH climatology (Pommrich et al., 2014)... */
01359 double aux00 = LIN(clim_oh_ps[ip],
01360                   clim_oh_var[isec][ilat][ip],
01361                   clim_oh_ps[ip + 1],
01362                   clim_oh_var[isec][ilat][ip + 1], p);
01363 double aux01 = LIN(clim_oh_ps[ip],
01364                   clim_oh_var[isec][ilat + 1][ip],
01365                   clim_oh_ps[ip + 1],
01366                   clim_oh_var[isec][ilat + 1][ip + 1], p);
01367 double aux10 = LIN(clim_oh_ps[ip],
01368                   clim_oh_var[isec + 1][ilat][ip],
01369                   clim_oh_ps[ip + 1],
01370                   clim_oh_var[isec + 1][ilat][ip + 1], p);
01371 double aux11 = LIN(clim_oh_ps[ip],
01372                   clim_oh_var[isec + 1][ilat + 1][ip],
01373                   clim_oh_ps[ip + 1],
01374                   clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01375 aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01376 aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01377 aux00 = LIN(clim_oh_secs[isec], aux00, clim_oh_secs[isec + 1], aux11, sec);
01378 return GSL_MAX(1e6 * aux00, 0.0);
01379 }

```

5.21.3.5 clim_tropo() double clim_tropo (
double t,
double lat)

Climatology of tropopause pressure.

Definition at line 1512 of file libtrac.c.

```

01514     {
01515
01516     /* Get seconds since begin of year... */
01517     double sec = FMOD(t, 365.25 * 86400.);
01518     while (sec < 0)
01519         sec += 365.25 * 86400.;
01520
01521     /* Get indices... */
01522     int isec = locate_irr(clim_tropo_secs, 12, sec);
01523     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01524
01525     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01526     double p0 = LIN(clim_tropo_lats[ilat],
01527                    clim_tropo_tps[isec][ilat],
01528                    clim_tropo_lats[ilat + 1],
01529                    clim_tropo_tps[isec][ilat + 1], lat);
01530     double p1 = LIN(clim_tropo_lats[ilat],
01531                    clim_tropo_tps[isec + 1][ilat],
01532                    clim_tropo_lats[ilat + 1],
01533                    clim_tropo_tps[isec + 1][ilat + 1], lat);
01534     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01535 }

```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 1539 of file libtrac.c.

```

01543     {
01544
01545     const int
01546     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01547     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01548
01549     /* Get day of year... */
01550     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01551         *doy = d0l[mon - 1] + day - 1;
01552     else
01553         *doy = d0[mon - 1] + day - 1;
01554 }

```

5.21.3.7 doy2day() void doy2day (

```

    int year,
    int doy,
    int * mon,
    int * day )
```

Get date from day of year.

Definition at line 1558 of file libtrac.c.

```

01562     {
01563
01564     const int
01565         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01566         d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01567
01568     int i;
01569
01570     /* Get month and day... */
01571     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01572         for (i = 11; i > 0; i--)
01573             if (d0l[i] <= doy)
01574                 break;
01575         *mon = i + 1;
01576         *day = doy - d0l[i] + 1;
01577     } else {
01578         for (i = 11; i > 0; i--)
01579             if (d0[i] <= doy)
01580                 break;
01581         *mon = i + 1;
01582         *day = doy - d0[i] + 1;
01583     }
01584 }
```

5.21.3.8 geo2cart() void geo2cart (

```

    double z,
    double lon,
    double lat,
    double * x )
```

Convert geolocation to Cartesian coordinates.

Definition at line 1588 of file libtrac.c.

```

01592     {
01593
01594     double radius = z + RE;
01595     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01596     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01597     x[2] = radius * sin(lat / 180. * M_PI);
01598 }
```

5.21.3.9 get_met() void get_met (

```

    ctl_t * ctl,
    double t,
    met_t ** met0,
    met_t ** met1 )
```

Get meteorological data for given time step.

Definition at line 1602 of file libtrac.c.

```

01606     {
01607
01608     static int init;
01609
01610     met_t *mets;
```

```

01611
01612 char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01613
01614 /* Set timer... */
01615 SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01616
01617 /* Init... */
01618 if (t == ctl->t_start || !init) {
01619     init = 1;
01620
01621     /* Read meteo data... */
01622     get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01623     if (!read_met(ctl, filename, *met0))
01624         ERRMSG("Cannot open file!");
01625
01626     get_met_help(t + 1.0 * ctl->direction, 1, ctl->metbase, ctl->dt_met,
01627                 filename);
01628     if (!read_met(ctl, filename, *met1))
01629         ERRMSG("Cannot open file!");
01630
01631     /* Update GPU... */
01632 #ifdef _OPENACC
01633     met_t *met0up = *met0;
01634     met_t *met1up = *met1;
01635 #pragma acc update device(met0up[:1],met1up[:1])
01636 #endif
01637
01638     /* Caching... */
01639     if (ctl->met_cache && t != ctl->t_stop) {
01640         get_met_help(t + 1.1 * ctl->dt_met * ctl->direction, ctl->direction,
01641                     ctl->metbase, ctl->dt_met, cachefile);
01642         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01643         LOG(1, "Caching: %s", cachefile);
01644         if (system(cmd) != 0)
01645             WARN("Caching command failed!");
01646     }
01647 }
01648
01649 /* Read new data for forward trajectories... */
01650 if (t > (*met1)->time) {
01651
01652     /* Pointer swap... */
01653     mets = *met1;
01654     *met1 = *met0;
01655     *met0 = mets;
01656
01657     /* Read new meteo data... */
01658     get_met_help(t, 1, ctl->metbase, ctl->dt_met, filename);
01659     if (!read_met(ctl, filename, *met1))
01660         ERRMSG("Cannot open file!");
01661
01662     /* Update GPU... */
01663 #ifdef _OPENACC
01664     met_t *met1up = *met1;
01665 #pragma acc update device(met1up[:1])
01666 #endif
01667
01668     /* Caching... */
01669     if (ctl->met_cache && t != ctl->t_stop) {
01670         get_met_help(t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met, cachefile);
01671         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01672         LOG(1, "Caching: %s", cachefile);
01673         if (system(cmd) != 0)
01674             WARN("Caching command failed!");
01675     }
01676 }
01677
01678 /* Read new data for backward trajectories... */
01679 if (t < (*met0)->time) {
01680
01681     /* Pointer swap... */
01682     mets = *met1;
01683     *met1 = *met0;
01684     *met0 = mets;
01685
01686     /* Read new meteo data... */
01687     get_met_help(t, -1, ctl->metbase, ctl->dt_met, filename);
01688     if (!read_met(ctl, filename, *met0))
01689         ERRMSG("Cannot open file!");
01690
01691     /* Update GPU... */
01692 #ifdef _OPENACC
01693     met_t *met0up = *met0;
01694 #pragma acc update device(met0up[:1])
01695 #endif
01696
01697     /* Caching... */

```

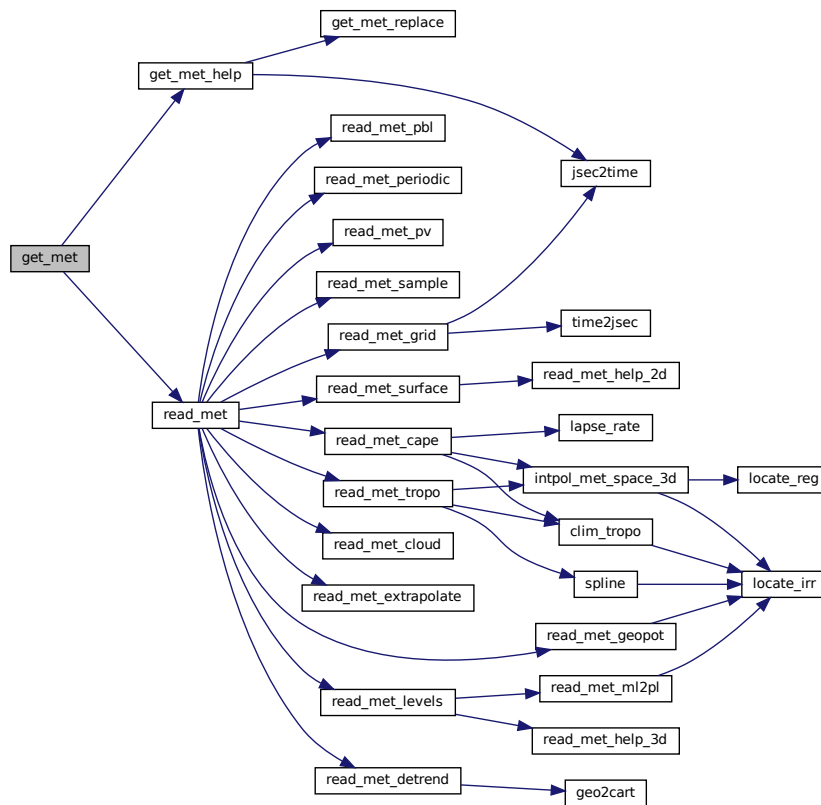


```

01698     if (ctl->met_cache && t != ctl->t_stop) {
01699         get_met_help(t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met, cachefile);
01700         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01701         LOG(1, "Caching: %s", cachefile);
01702         if (system(cmd) != 0)
01703             WARN("Caching command failed!");
01704     }
01705 }
01706
01707 /* Check that grids are consistent... */
01708 if ((*met0)->nx != (*met1)->nx
01709     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01710     ERRMSG("Meteo grid dimensions do not match!");
01711 for (int ix = 0; ix < (*met0)->nx; ix++)
01712     if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01713         ERRMSG("Meteo grid longitudes do not match!");
01714 for (int iy = 0; iy < (*met0)->ny; iy++)
01715     if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01716         ERRMSG("Meteo grid latitudes do not match!");
01717 for (int ip = 0; ip < (*met0)->np; ip++)
01718     if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01719         ERRMSG("Meteo grid pressure levels do not match!");
01720 }

```

Here is the call graph for this function:



5.21.3.10 get_met_help() void get_met_help (

```

    double t,
    int direct,
    char * metbase,
    double dt_met,
    char * filename )

```

Get meteorological data for time step.

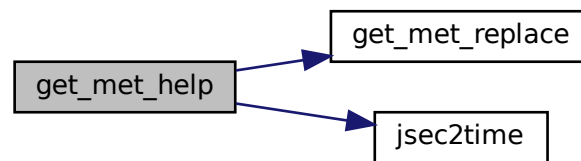
Definition at line 1724 of file [libtrac.c](#).

```

01729     {
01730
01731     char repl[LEN];
01732
01733     double t6, r;
01734
01735     int year, mon, day, hour, min, sec;
01736
01737     /* Round time to fixed intervals... */
01738     if (direct == -1)
01739         t6 = floor(t / dt_met) * dt_met;
01740     else
01741         t6 = ceil(t / dt_met) * dt_met;
01742
01743     /* Decode time... */
01744     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01745
01746     /* Set filename... */
01747     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01748     sprintf(repl, "%d", year);
01749     get_met_replace(filename, "YYYY", repl);
01750     sprintf(repl, "%02d", mon);
01751     get_met_replace(filename, "MM", repl);
01752     sprintf(repl, "%02d", day);
01753     get_met_replace(filename, "DD", repl);
01754     sprintf(repl, "%02d", hour);
01755     get_met_replace(filename, "HH", repl);
01756 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1760 of file [libtrac.c](#).

```

01763     {
01764
01765     char buffer[LEN], *ch;
01766
01767     /* Iterate... */
01768     for (int i = 0; i < 3; i++) {
01769
01770         /* Replace sub-string... */
01771         if (!(ch = strstr(orig, search)))
01772             return;
01773         strncpy(buffer, orig, (size_t) (ch - orig));
01774         buffer[ch - orig] = 0;

```

```

01775     sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01776     orig[0] = 0;
01777     strcpy(orig, buffer);
01778 }
01779 }

```

5.21.3.12 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 meteorological data.

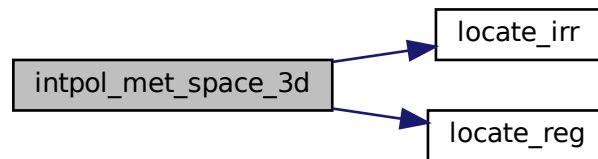
Definition at line 1783 of file libtrac.c.

```

01792     {
01793
01794     /* Initialize interpolation... */
01795     if (init) {
01796
01797     /* Check longitude... */
01798     if (met->lon[met->nx - 1] > 180 && lon < 0)
01799         lon += 360;
01800
01801     /* Get interpolation indices... */
01802     ci[0] = locate_irr(met->p, met->np, p);
01803     ci[1] = locate_reg(met->lon, met->nx, lon);
01804     ci[2] = locate_reg(met->lat, met->ny, lat);
01805
01806     /* Get interpolation weights... */
01807     cw[0] = (met->p[ci[0] + 1] - p)
01808         / (met->p[ci[0] + 1] - met->p[ci[0]]);
01809     cw[1] = (met->lon[ci[1] + 1] - lon)
01810         / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01811     cw[2] = (met->lat[ci[2] + 1] - lat)
01812         / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01813     }
01814
01815     /* Interpolate vertically... */
01816     double aux00 =
01817         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01818         + array[ci[1]][ci[2]][ci[0] + 1];
01819     double aux01 =
01820         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01821             array[ci[1]][ci[2] + 1][ci[0] + 1])
01822         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01823     double aux10 =
01824         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01825             array[ci[1] + 1][ci[2]][ci[0] + 1])
01826         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01827     double aux11 =
01828         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01829             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01830         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01831
01832     /* Interpolate horizontally... */
01833     aux00 = cw[2] * (aux00 - aux01) + aux01;
01834     aux11 = cw[2] * (aux10 - aux11) + aux11;
01835     *var = cw[1] * (aux00 - aux11) + aux11;
01836 }

```

Here is the call graph for this function:



5.21.3.13 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 meteorological data.

Definition at line 1841 of file libtrac.c.

```

1849     {
1850
1851     /* Initialize interpolation... */
1852     if (init) {
1853
1854     /* Check longitude... */
1855     if (met->lon[met->nx - 1] > 180 && lon < 0)
1856         lon += 360;
1857
1858     /* Get interpolation indices... */
1859     ci[1] = locate_reg(met->lon, met->nx, lon);
1860     ci[2] = locate_reg(met->lat, met->ny, lat);
1861
1862     /* Get interpolation weights... */
1863     cw[1] = (met->lon[ci[1] + 1] - lon)
1864             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
1865     cw[2] = (met->lat[ci[2] + 1] - lat)
1866             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
1867     }
1868
1869     /* Set variables... */
1870     double aux00 = array[ci[1]][ci[2]];
1871     double aux01 = array[ci[1]][ci[2] + 1];
1872     double aux10 = array[ci[1] + 1][ci[2]];
1873     double aux11 = array[ci[1] + 1][ci[2] + 1];
1874
1875     /* Interpolate horizontally... */
1876     if (isfinite(aux00) && isfinite(aux01)
1877         && isfinite(aux10) && isfinite(aux11)) {
1878         aux00 = cw[2] * (aux00 - aux01) + aux01;
1879         aux11 = cw[2] * (aux10 - aux11) + aux11;
1880         *var = cw[1] * (aux00 - aux11) + aux11;
1881     } else {
1882         if (cw[2] < 0.5) {
1883             if (cw[1] < 0.5)
1884                 *var = aux11;
1885             else
1886                 *var = aux01;
1887         } else {

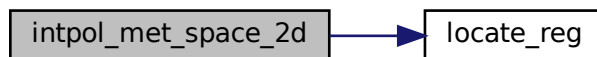
```

```

01888     if (cw[1] < 0.5)
01889         *var = aux10;
01890     else
01891         *var = aux00;
01892 }
01893 }
01894 }

```

Here is the call graph for this function:



5.21.3.14 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 )

```

Temporal interpolation of meteorological data.

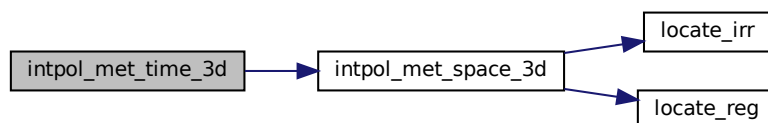
Definition at line 1898 of file libtrac.c.

```

01910     {
01911
01912     double var0, var1, wt;
01913
01914     /* Spatial interpolation... */
01915     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01916     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01917
01918     /* Get weighting factor... */
01919     wt = (met1->time - ts) / (met1->time - met0->time);
01920
01921     /* Interpolate... */
01922     *var = wt * (var0 - var1) + var1;
01923 }

```

Here is the call graph for this function:



```

5.21.3.15 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 meteorological data.

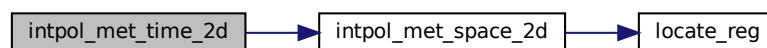
Definition at line 1927 of file libtrac.c.

```

01938     {
01939
01940     double var0, var1, wt;
01941
01942     /* Spatial interpolation... */
01943     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01944     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01945
01946     /* Get weighting factor... */
01947     wt = (met1->time - ts) / (met1->time - met0->time);
01948
01949     /* Interpolate... */
01950     if (isfinite(var0) && isfinite(var1))
01951         *var = wt * (var0 - var1) + var1;
01952     else if (wt < 0.5)
01953         *var = var1;
01954     else
01955         *var = var0;
01956 }

```

Here is the call graph for this function:



```

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

```

Convert seconds to date.

Definition at line 1960 of file [libtrac.c](#).

```

01968         {
01969
01970     struct tm t0, *t1;
01971
01972     t0.tm_year = 100;
01973     t0.tm_mon = 0;
01974     t0.tm_mday = 1;
01975     t0.tm_hour = 0;
01976     t0.tm_min = 0;
01977     t0.tm_sec = 0;
01978
01979     time_t jsec0 = (time_t) jsec + timegm(&t0);
01980     t1 = gmtime(&jsec0);
01981
01982     *year = t1->tm_year + 1900;
01983     *mon = t1->tm_mon + 1;
01984     *day = t1->tm_mday;
01985     *hour = t1->tm_hour;
01986     *min = t1->tm_min;
01987     *sec = t1->tm_sec;
01988     *remain = jsec - floor(jsec);
01989 }
```

5.21.3.17 lapse_rate() double lapse_rate (
 double t,
 double h2o)

Calculate moist adiabatic lapse rate.

Definition at line 1993 of file [libtrac.c](#).

```

01995     {
01996
01997     /*
01998         Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01999         and water vapor volume mixing ratio [1].
02000
02001         Reference: https://en.wikipedia.org/wiki/Lapse\_rate
02002     */
02003
02004     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
02005
02006     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
02007 }
```

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

Find array index for irregular grid.

Definition at line 2011 of file [libtrac.c](#).

```

02014     {
02015
02016     int ilo = 0;
02017     int ihi = n - 1;
02018     int i = (ihi + ilo) >> 1;
02019
02020     if (xx[i] < xx[i + 1])
02021         while (ihi > ilo + 1) {
02022             i = (ihi + ilo) >> 1;
02023             if (xx[i] > x)
02024                 ihi = i;
02025             else
02026                 ilo = i;
02027         } else
02028         while (ihi > ilo + 1) {
02029             i = (ihi + ilo) >> 1;
```

```

02030         if (xx[i] <= x)
02031             ihi = i;
02032         else
02033             ilo = i;
02034     }
02035
02036     return ilo;
02037 }

```

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

Find array index for regular grid.

Definition at line 2041 of file libtrac.c.

```

02044     {
02045
02046         /* Calculate index... */
02047         int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
02048
02049         /* Check range... */
02050         if (i < 0)
02051             return 0;
02052         else if (i > n - 2)
02053             return n - 2;
02054         else
02055             return i;
02056 }

```

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

Calculate NAT existence temperature.

Definition at line 2060 of file libtrac.c.

```

02063     {
02064
02065         /* Check water vapor vmr... */
02066         h2o = GSL_MAX(h2o, 0.1e-6);
02067
02068         /* Calculate T_NAT... */
02069         double p_hno3 = hno3 * p / 1.333224;
02070         double p_h2o = h2o * p / 1.333224;
02071         double a = 0.009179 - 0.00088 * log10(p_h2o);
02072         double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
02073         double c = -11397.0 / a;
02074         double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
02075         double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
02076         if (x2 > 0)
02077             tnat = x2;
02078
02079         return tnat;
02080 }

```


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

Read atmospheric data.

Definition at line 2084 of file libtrac.c.

```

2087     {
2088
2089     FILE *in;
2090
2091     double t0;
2092
2093     int dimid, ncid, varid;
2094
2095     size_t nparts;
2096
2097     /* Set timer... */
2098     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
2099
2100     /* Init... */
2101     atm->np = 0;
2102
2103     /* Write info... */
2104     LOG(1, "Read atmospheric data: %s", filename);
2105
2106     /* Read ASCII data... */
2107     if (ctl->atm_type == 0) {
2108
2109         /* Open file... */
2110         if (!(in = fopen(filename, "r"))) {
2111             WARN("File not found!");
2112             return 0;
2113         }
2114
2115         /* Read line... */
2116         char line[LEN];
2117         while (fgets(line, LEN, in)) {
2118
2119             /* Read data... */
2120             char *tok;
2121             TOK(line, tok, "%lg", atm->time[atm->np]);
2122             TOK(NULL, tok, "%lg", atm->p[atm->np]);
2123             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
2124             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
2125             for (int iq = 0; iq < ctl->nq; iq++)
2126                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
2127
2128             /* Convert altitude to pressure... */
2129             atm->p[atm->np] = P(atm->p[atm->np]);
2130
2131             /* Increment data point counter... */
2132             if ((++atm->np) > NP)
2133                 ERRMSG("Too many data points!");
2134         }
2135
2136         /* Close file... */
2137         fclose(in);
2138     }
2139
2140     /* Read binary data... */
2141     else if (ctl->atm_type == 1) {
2142
2143         /* Open file... */
2144         if (!(in = fopen(filename, "r")))
2145             return 0;
2146
2147         /* Read data... */
2148         FREAD(&atm->np, int,
2149             1,
2150             in);
2151         FREAD(atm->time, double,
2152             (size_t) atm->np,
2153             in);
2154         FREAD(atm->p, double,
2155             (size_t) atm->np,
2156             in);
2157         FREAD(atm->lon, double,
2158             (size_t) atm->np,
2159             in);
2160         FREAD(atm->lat, double,
2161             (size_t) atm->np,
2162             in);

```

```

02163     for (int iq = 0; iq < ctl->nq; iq++)
02164         FREAD(atm->q[iq], double,
02165             (size_t) atm->np,
02166             in);
02167
02168     /* Close file... */
02169     fclose(in);
02170 }
02171
02172 /* Read netCDF data... */
02173 else if (ctl->atm_type == 2) {
02174
02175     /* Open file... */
02176     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02177         return 0;
02178
02179     /* Get dimensions... */
02180     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02181     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02182     atm->np = (int) nparts;
02183     if (atm->np > NP)
02184         ERRMSG("Too many particles!");
02185
02186     /* Get time... */
02187     NC(nc_inq_varid(ncid, "time", &varid));
02188     NC(nc_get_var_double(ncid, varid, &t0));
02189     for (int ip = 0; ip < atm->np; ip++)
02190         atm->time[ip] = t0;
02191
02192     /* Read geolocations... */
02193     NC(nc_inq_varid(ncid, "PRESS", &varid));
02194     NC(nc_get_var_double(ncid, varid, atm->p));
02195     NC(nc_inq_varid(ncid, "LON", &varid));
02196     NC(nc_get_var_double(ncid, varid, atm->lon));
02197     NC(nc_inq_varid(ncid, "LAT", &varid));
02198     NC(nc_get_var_double(ncid, varid, atm->lat));
02199
02200     /* Read variables... */
02201     if (ctl->qnt_p >= 0)
02202         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02203             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02204     if (ctl->qnt_t >= 0)
02205         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02206             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02207     if (ctl->qnt_u >= 0)
02208         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02209             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02210     if (ctl->qnt_v >= 0)
02211         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02212             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02213     if (ctl->qnt_w >= 0)
02214         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02215             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02216     if (ctl->qnt_h2o >= 0)
02217         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02218             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02219     if (ctl->qnt_o3 >= 0)
02220         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02221             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02222     if (ctl->qnt_theta >= 0)
02223         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02224             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02225     if (ctl->qnt_pv >= 0)
02226         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02227             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02228
02229     /* Check data... */
02230     for (int ip = 0; ip < atm->np; ip++)
02231         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02232             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02233             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02234             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02235             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10) {
02236             atm->time[ip] = GSL_NAN;
02237             atm->p[ip] = GSL_NAN;
02238             atm->lon[ip] = GSL_NAN;
02239             atm->lat[ip] = GSL_NAN;
02240             for (int iq = 0; iq < ctl->nq; iq++)
02241                 atm->q[iq][ip] = GSL_NAN;
02242         } else {
02243             if (ctl->qnt_h2o >= 0)
02244                 atm->q[ctl->qnt_h2o][ip] *= 1.608;
02245             if (ctl->qnt_pv >= 0)
02246                 atm->q[ctl->qnt_pv][ip] *= 1e6;
02247             if (atm->lon[ip] > 180)
02248                 atm->lon[ip] -= 360;
02249         }

```

```

02250
02251     /* Close file... */
02252     NC(nc_close(ncid));
02253 }
02254
02255 /* Error... */
02256 else
02257     ERRMSG("Atmospheric data type not supported!");
02258
02259 /* Check number of points... */
02260 if (atm->np < 1)
02261     ERRMSG("Can not read any data!");
02262
02263 /* Write info... */
02264 double mini, maxi;
02265 LOG(2, "Number of particles: %d", atm->np);
02266 gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
02267 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
02268 gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
02269 LOG(2, "Altitude range: %g ... %g km", Z(mini), Z(maxi));
02270 LOG(2, "Pressure range: %g ... %g hPa", mini, maxi);
02271 gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
02272 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
02273 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
02274 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
02275 for (int iq = 0; iq < ctl->nq; iq++) {
02276     char msg[LEN];
02277     sprintf(msg, "Quantity %s range: %s ... %s %s",
02278             ctl->qnt_name[iq], ctl->qnt_format[iq],
02279             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
02280     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
02281     LOG(2, msg, mini, maxi);
02282 }
02283
02284 /* Return success... */
02285 return 1;
02286 }

```

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

Read control parameters.

Definition at line 2290 of file libtrac.c.

```

02294     {
02295
02296     /* Set timer... */
02297     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02298
02299     /* Write info... */
02300     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02301           "(executable: %s | version: %s | compiled: %s, %s)\n",
02302         argv[0], VERSION, __DATE__, __TIME__);
02303
02304     /* Initialize quantity indices... */
02305     ctl->qnt_ens = -1;
02306     ctl->qnt_stat = -1;
02307     ctl->qnt_m = -1;
02308     ctl->qnt_vmr = -1;
02309     ctl->qnt_r = -1;
02310     ctl->qnt_rho = -1;
02311     ctl->qnt_ps = -1;
02312     ctl->qnt_ts = -1;
02313     ctl->qnt_zs = -1;
02314     ctl->qnt_us = -1;
02315     ctl->qnt_vs = -1;
02316     ctl->qnt_pbl = -1;
02317     ctl->qnt_pt = -1;
02318     ctl->qnt_tt = -1;
02319     ctl->qnt_zt = -1;
02320     ctl->qnt_h2ot = -1;
02321     ctl->qnt_z = -1;
02322     ctl->qnt_p = -1;
02323     ctl->qnt_t = -1;

```

```

02324     ctl->qnt_u = -1;
02325     ctl->qnt_v = -1;
02326     ctl->qnt_w = -1;
02327     ctl->qnt_h2o = -1;
02328     ctl->qnt_o3 = -1;
02329     ctl->qnt_lwc = -1;
02330     ctl->qnt_iwc = -1;
02331     ctl->qnt_pct = -1;
02332     ctl->qnt_pcb = -1;
02333     ctl->qnt_cl = -1;
02334     ctl->qnt_plcl = -1;
02335     ctl->qnt_plfc = -1;
02336     ctl->qnt_pel = -1;
02337     ctl->qnt_cape = -1;
02338     ctl->qnt_cin = -1;
02339     ctl->qnt_hno3 = -1;
02340     ctl->qnt_oh = -1;
02341     ctl->qnt_psat = -1;
02342     ctl->qnt_psice = -1;
02343     ctl->qnt_pw = -1;
02344     ctl->qnt_sh = -1;
02345     ctl->qnt_rh = -1;
02346     ctl->qnt_rhice = -1;
02347     ctl->qnt_theta = -1;
02348     ctl->qnt_zeta = -1;
02349     ctl->qnt_tvirt = -1;
02350     ctl->qnt_lapse = -1;
02351     ctl->qnt_vh = -1;
02352     ctl->qnt_vz = -1;
02353     ctl->qnt_pv = -1;
02354     ctl->qnt_tdew = -1;
02355     ctl->qnt_tice = -1;
02356     ctl->qnt_tsts = -1;
02357     ctl->qnt_tnat = -1;
02358
02359     /* Read quantities... */
02360     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02361     if (ctl->nq > NQ)
02362         ERRMSG("Too many quantities!");
02363     for (int iq = 0; iq < ctl->nq; iq++) {
02364
02365         /* Read quantity name and format... */
02366         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02367         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02368             ctl->qnt_format[iq]);
02369
02370         /* Try to identify quantity... */
02371         SET_QNT(qnt_ens, "ens", "-")
02372         SET_QNT(qnt_stat, "stat", "-")
02373         SET_QNT(qnt_m, "m", "kg")
02374         SET_QNT(qnt_vmr, "vmr", "ppv")
02375         SET_QNT(qnt_x, "x", "microns")
02376         SET_QNT(qnt_rho, "rho", "kg/m^3")
02377         SET_QNT(qnt_ps, "ps", "hPa")
02378         SET_QNT(qnt_ts, "ts", "K")
02379         SET_QNT(qnt_zs, "zs", "km")
02380         SET_QNT(qnt_us, "us", "m/s")
02381         SET_QNT(qnt_vs, "vs", "m/s")
02382         SET_QNT(qnt_pbl, "pbl", "hPa")
02383         SET_QNT(qnt_pt, "pt", "hPa")
02384         SET_QNT(qnt_tt, "tt", "K")
02385         SET_QNT(qnt_zt, "zt", "km")
02386         SET_QNT(qnt_h2ot, "h2ot", "ppv")
02387         SET_QNT(qnt_z, "z", "km")
02388         SET_QNT(qnt_p, "p", "hPa")
02389         SET_QNT(qnt_t, "t", "K")
02390         SET_QNT(qnt_u, "u", "m/s")
02391         SET_QNT(qnt_v, "v", "m/s")
02392         SET_QNT(qnt_w, "w", "hPa/s")
02393         SET_QNT(qnt_h2o, "h2o", "ppv")
02394         SET_QNT(qnt_o3, "o3", "ppv")
02395         SET_QNT(qnt_lwc, "lwc", "kg/kg")
02396         SET_QNT(qnt_iwc, "iwc", "kg/kg")
02397         SET_QNT(qnt_pct, "pct", "hPa")
02398         SET_QNT(qnt_pcb, "pcb", "hPa")
02399         SET_QNT(qnt_cl, "cl", "kg/m^2")
02400         SET_QNT(qnt_plcl, "plcl", "hPa")
02401         SET_QNT(qnt_plfc, "plfc", "hPa")
02402         SET_QNT(qnt_pel, "pel", "hPa")
02403         SET_QNT(qnt_cape, "cape", "J/kg")
02404         SET_QNT(qnt_cin, "cin", "J/kg")
02405         SET_QNT(qnt_hno3, "hno3", "ppv")
02406         SET_QNT(qnt_oh, "oh", "molec/cm^3")
02407         SET_QNT(qnt_psat, "psat", "hPa")
02408         SET_QNT(qnt_psice, "psice", "hPa")
02409         SET_QNT(qnt_pw, "pw", "hPa")
02410         SET_QNT(qnt_sh, "sh", "kg/kg")

```

```

02411     SET_QNT(qnt_rh, "rh", "%")
02412     SET_QNT(qnt_rhice, "rhice", "%")
02413     SET_QNT(qnt_theta, "theta", "K")
02414     SET_QNT(qnt_zeta, "zeta", "K")
02415     SET_QNT(qnt_tvirt, "tvirt", "K")
02416     SET_QNT(qnt_lapse, "lapse", "K/km")
02417     SET_QNT(qnt_vh, "vh", "m/s")
02418     SET_QNT(qnt_vz, "vz", "m/s")
02419     SET_QNT(qnt_pv, "pv", "PVU")
02420     SET_QNT(qnt_tdew, "tdew", "K")
02421     SET_QNT(qnt_tice, "tice", "K")
02422     SET_QNT(qnt_tsts, "tsts", "K")
02423     SET_QNT(qnt_tnat, "tnat", "K")
02424     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02425 }
02426
02427 /* netCDF I/O parameters... */
02428 ctl->chunkszhint =
02429     (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02430         NULL);
02431 ctl->read_mode =
02432     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02433
02434 /* Time steps of simulation... */
02435 ctl->direction =
02436     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02437 if (ctl->direction != -1 && ctl->direction != 1)
02438     ERRMSG("Set DIRECTION to -1 or 1!");
02439 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02440 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02441
02442 /* Meteorological data... */
02443 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02444 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02445 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02446 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02447 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02448 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02449     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02450 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02451 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02452 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02453 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02454     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02455 ctl->met_detrend =
02456     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02457 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02458 if (ctl->met_np > EP)
02459     ERRMSG("Too many levels!");
02460 for (int ip = 0; ip < ctl->met_np; ip++)
02461     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02462 ctl->met_geopot_sx
02463     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02464 ctl->met_geopot_sy
02465     = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02466 ctl->met_tropo =
02467     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02468 if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02469     ERRMSG("Set MET_TROPO = 0 ... 5!");
02470 ctl->met_tropo_lapse =
02471     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02472 ctl->met_tropo_nlev =
02473     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02474 ctl->met_tropo_lapse_sep =
02475     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02476 ctl->met_tropo_nlev_sep =
02477     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02478         NULL);
02479 ctl->met_tropo_pv =
02480     scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02481 ctl->met_tropo_theta =
02482     scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02483 ctl->met_tropo_spline =
02484     (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02485 ctl->met_cloud =
02486     (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02487 if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02488     ERRMSG("Set MET_CLOUD = 0 ... 3!");
02489 ctl->met_dt_out =
02490     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02491 ctl->met_cache =
02492     (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02493
02494 /* Isosurface parameters... */
02495 ctl->isosurf =
02496     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02497 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

```

```

02498
02499 /* Advection parameters... */
02500 ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "0", NULL);
02501 ctl->reflect =
02502     (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02503
02504 /* Diffusion parameters... */
02505 ctl->turb_dx_trop =
02506     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02507 ctl->turb_dx_strat =
02508     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02509 ctl->turb_dz_trop =
02510     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02511 ctl->turb_dz_strat =
02512     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02513 ctl->turb_mesox =
02514     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02515 ctl->turb_mesoz =
02516     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02517
02518 /* Convection... */
02519 ctl->conv_cape
02520     = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02521 ctl->conv_cin
02522     = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02523 ctl->conv_wmax
02524     = scan_ctl(filename, argc, argv, "CONV_WMAX", -1, "-999", NULL);
02525 ctl->conv_wcape
02526     = (int) scan_ctl(filename, argc, argv, "CONV_WCAPE", -1, "0", NULL);
02527 ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02528 ctl->conv_mix_bot
02529     = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02530 ctl->conv_mix_top
02531     = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02532
02533 /* Boundary conditions... */
02534 ctl->bound_mass =
02535     scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02536 ctl->bound_vmr =
02537     scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02538 ctl->bound_lat0 =
02539     scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02540 ctl->bound_lat1 =
02541     scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02542 ctl->bound_p0 =
02543     scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02544 ctl->bound_p1 =
02545     scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02546 ctl->bound_dps =
02547     scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02548
02549 /* Species parameters... */
02550 scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02551 if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02552     ctl->molmass = 120.907;
02553     ctl->wet_depo[2] = ctl->wet_depo[6] = 3e-5;
02554     ctl->wet_depo[3] = ctl->wet_depo[7] = 3500.0;
02555 } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02556     ctl->molmass = 137.359;
02557     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.1e-4;
02558     ctl->wet_depo[3] = ctl->wet_depo[7] = 3300.0;
02559 } else if (strcasecmp(ctl->species, "CH4") == 0) {
02560     ctl->molmass = 16.043;
02561     ctl->oh_chem_reaction = 2;
02562     ctl->oh_chem[0] = 2.45e-12;
02563     ctl->oh_chem[1] = 1775;
02564     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.4e-5;
02565     ctl->wet_depo[3] = ctl->wet_depo[7] = 1600.0;
02566 } else if (strcasecmp(ctl->species, "CO") == 0) {
02567     ctl->molmass = 28.01;
02568     ctl->oh_chem_reaction = 3;
02569     ctl->oh_chem[0] = 6.9e-33;
02570     ctl->oh_chem[1] = 2.1;
02571     ctl->oh_chem[2] = 1.1e-12;
02572     ctl->oh_chem[3] = -1.3;
02573     ctl->wet_depo[2] = ctl->wet_depo[6] = 9.7e-6;
02574     ctl->wet_depo[3] = ctl->wet_depo[7] = 1300.0;
02575 } else if (strcasecmp(ctl->species, "CO2") == 0) {
02576     ctl->molmass = 44.009;
02577     ctl->wet_depo[2] = ctl->wet_depo[6] = 3.3e-4;
02578     ctl->wet_depo[3] = ctl->wet_depo[7] = 2400.0;
02579 } else if (strcasecmp(ctl->species, "N2O") == 0) {
02580     ctl->molmass = 44.013;
02581     ctl->wet_depo[2] = ctl->wet_depo[6] = 2.4e-4;
02582     ctl->wet_depo[3] = ctl->wet_depo[7] = 2600.;
02583 } else if (strcasecmp(ctl->species, "NH3") == 0) {
02584     ctl->molmass = 17.031;

```

```

02585     ctl->oh_chem_reaction = 2;
02586     ctl->oh_chem[0] = 1.7e-12;
02587     ctl->oh_chem[1] = 710;
02588     ctl->wet_depo[2] = ctl->wet_depo[6] = 5.9e-1;
02589     ctl->wet_depo[3] = ctl->wet_depo[7] = 4200.0;
02590 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02591     ctl->molmass = 63.012;
02592     ctl->wet_depo[2] = ctl->wet_depo[6] = 2.1e3;
02593     ctl->wet_depo[3] = ctl->wet_depo[7] = 8700.0;
02594 } else if (strcasecmp(ctl->species, "NO") == 0) {
02595     ctl->molmass = 30.006;
02596     ctl->oh_chem_reaction = 3;
02597     ctl->oh_chem[0] = 7.1e-31;
02598     ctl->oh_chem[1] = 2.6;
02599     ctl->oh_chem[2] = 3.6e-11;
02600     ctl->oh_chem[3] = 0.1;
02601     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.9e-5;
02602     ctl->wet_depo[3] = ctl->wet_depo[7] = 1600.0;
02603 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02604     ctl->molmass = 46.005;
02605     ctl->oh_chem_reaction = 3;
02606     ctl->oh_chem[0] = 1.8e-30;
02607     ctl->oh_chem[1] = 3.0;
02608     ctl->oh_chem[2] = 2.8e-11;
02609     ctl->oh_chem[3] = 0.0;
02610     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.2e-4;
02611     ctl->wet_depo[3] = ctl->wet_depo[7] = 2400.0;
02612 } else if (strcasecmp(ctl->species, "O3") == 0) {
02613     ctl->molmass = 47.997;
02614     ctl->oh_chem_reaction = 2;
02615     ctl->oh_chem[0] = 1.7e-12;
02616     ctl->oh_chem[1] = 940;
02617     ctl->wet_depo[2] = ctl->wet_depo[6] = 1e-4;
02618     ctl->wet_depo[3] = ctl->wet_depo[7] = 2800.0;
02619 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02620     ctl->molmass = 146.048;
02621     ctl->wet_depo[2] = ctl->wet_depo[6] = 2.4e-6;
02622     ctl->wet_depo[3] = ctl->wet_depo[7] = 3100.0;
02623 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02624     ctl->molmass = 64.066;
02625     ctl->oh_chem_reaction = 3;
02626     ctl->oh_chem[0] = 2.9e-31;
02627     ctl->oh_chem[1] = 4.1;
02628     ctl->oh_chem[2] = 1.7e-12;
02629     ctl->oh_chem[3] = -0.2;
02630     ctl->wet_depo[2] = ctl->wet_depo[6] = 1.3e-2;
02631     ctl->wet_depo[3] = ctl->wet_depo[7] = 2900.0;
02632 } else {
02633     ctl->molmass =
02634         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02635     ctl->oh_chem_reaction =
02636         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02637     for (int ip = 0; ip < 4; ip++)
02638         ctl->oh_chem[ip] =
02639             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02640     for (int ip = 0; ip < 1; ip++)
02641         ctl->dry_depo[ip] =
02642             scan_ctl(filename, argc, argv, "DRY_DEPO", ip, "0", NULL);
02643     for (int ip = 0; ip < 8; ip++)
02644         ctl->wet_depo[ip] =
02645             scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02646 }
02647 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02648 ctl->tdec_strat =
02649     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02650
02651 /* PSC analysis... */
02652 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02653 ctl->psc_hno3 =
02654     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02655
02656 /* Output of atmospheric data... */
02657 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02658 scan_ctl(filename, argc, argv, "ATM_GPFIL", -1, "-", ctl->atm_gpfile);
02659 ctl->atm_dt_out =
02660     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02661 ctl->atm_filter =
02662     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02663 ctl->atm_stride =
02664     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02665 ctl->atm_type =
02666     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02667
02668 /* Output of CSI data... */
02669 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02670 ctl->csi_dt_out =
02671     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);

```

```

02672 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02673 ctl->csi_obsmin =
02674     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02675 ctl->csi_modmin =
02676     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02677 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02678 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02679 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02680 ctl->csi_lon0 =
02681     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02682 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02683 ctl->csi_nx =
02684     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02685 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02686 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02687 ctl->csi_ny =
02688     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02689
02690 /* Output of ensemble data... */
02691 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02692
02693 /* Output of grid data... */
02694 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02695     ctl->grid_basename);
02696 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02697 ctl->grid_dt_out =
02698     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02699 ctl->grid_sparse =
02700     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02701 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02702 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02703 ctl->grid_nz =
02704     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02705 ctl->grid_lon0 =
02706     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02707 ctl->grid_lon1 =
02708     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02709 ctl->grid_nx =
02710     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02711 ctl->grid_lat0 =
02712     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02713 ctl->grid_lat1 =
02714     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02715 ctl->grid_ny =
02716     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02717
02718 /* Output of profile data... */
02719 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02720     ctl->prof_basename);
02721 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02722 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02723 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02724 ctl->prof_nz =
02725     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02726 ctl->prof_lon0 =
02727     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02728 ctl->prof_lon1 =
02729     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02730 ctl->prof_nx =
02731     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02732 ctl->prof_lat0 =
02733     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02734 ctl->prof_lat1 =
02735     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02736 ctl->prof_ny =
02737     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02738
02739 /* Output of sample data... */
02740 scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02741     ctl->sample_basename);
02742 scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02743     ctl->sample_obsfile);
02744 ctl->sample_dx =
02745     scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02746 ctl->sample_dz =
02747     scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02748
02749 /* Output of station data... */
02750 scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02751     ctl->stat_basename);
02752 ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02753 ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02754 ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02755 ctl->stat_t0 =
02756     scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02757 ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02758 }

```


Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 2762 of file libtrac.c.

```

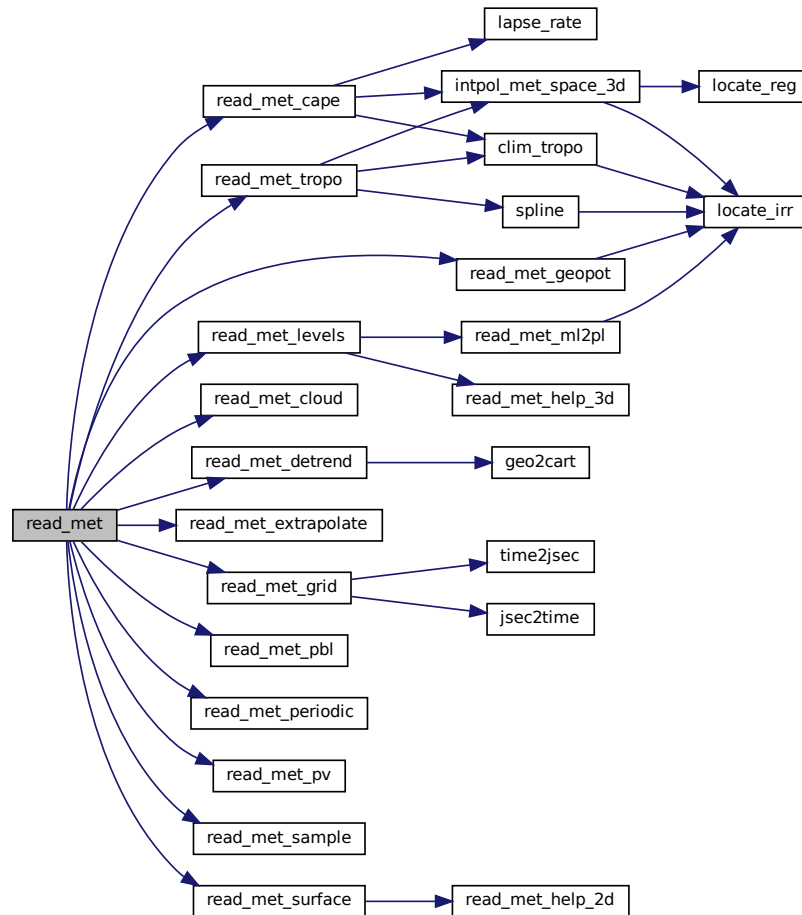
02765     {
02766
02767     int ncid;
02768
02769     /* Write info... */
02770     LOG(1, "Read meteorological data: %s", filename);
02771
02772     /* Open netCDF file... */
02773     if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02774         NC_NOERR) {
02775         WARN("File not found!");
02776         return 0;
02777     }
02778
02779     /* Read coordinates of meteorological data... */
02780     read_met_grid(filename, ncid, ctl, met);
02781
02782     /* Read meteo data on vertical levels... */
02783     read_met_levels(ncid, ctl, met);
02784
02785     /* Extrapolate data for lower boundary... */
02786     read_met_extrapolate(met);
02787
02788     /* Read surface data... */
02789     read_met_surface(ncid, met);
02790
02791     /* Create periodic boundary conditions... */
02792     read_met_periodic(met);
02793
02794     /* Downsampling... */
02795     read_met_sample(ctl, met);
02796
02797     /* Calculate geopotential heights... */
02798     read_met_geopot(ctl, met);
02799
02800     /* Calculate potential vorticity... */
02801     read_met_pv(met);
02802
02803     /* Calculate boundary layer data... */
02804     read_met_pbl(met);
02805
02806     /* Calculate tropopause data... */
02807     read_met_tropo(ctl, met);
02808
02809     /* Calculate cloud properties... */
02810     read_met_cloud(met);
02811
02812     /* Calculate convective available potential energy... */
02813     read_met_cape(met);
02814
02815     /* Detrending... */
  
```

```

02816  read_met_detrend(ctl, met);
02817
02818  /* Close file... */
02819  NC(nc_close(ncid));
02820
02821  /* Return success... */
02822  return 1;
02823 }

```

Here is the call graph for this function:



5.21.3.24 read_met_cape() void read_met_cape (
 met_t * met)

Calculate convective available potential energy.

Definition at line 2827 of file libtrac.c.

```

02828  {
02829
02830  /* Set timer... */
02831  SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
02832  LOG(2, "Calculate CAPE...");
02833
02834  /* Vertical spacing (about 100 m)... */

```

```

02835     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
02836
02837     /* Loop over columns... */
02838     #pragma omp parallel for default(shared) collapse(2)
02839     for (int ix = 0; ix < met->nx; ix++)
02840         for (int iy = 0; iy < met->ny; iy++) {
02841
02842             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
02843             int n = 0;
02844             double h2o = 0, t, theta = 0;
02845             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
02846             double ptop = pbot - 50.;
02847             for (int ip = 0; ip < met->np; ip++) {
02848                 if (met->p[ip] <= pbot) {
02849                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
02850                     h2o += met->h2o[ix][iy][ip];
02851                     n++;
02852                 }
02853                 if (met->p[ip] < ptop && n > 0)
02854                     break;
02855             }
02856             theta /= n;
02857             h2o /= n;
02858
02859             /* Cannot compute anything if water vapor is missing... */
02860             met->plcl[ix][iy] = GSL_NAN;
02861             met->plfc[ix][iy] = GSL_NAN;
02862             met->pel[ix][iy] = GSL_NAN;
02863             met->cape[ix][iy] = GSL_NAN;
02864             met->cin[ix][iy] = GSL_NAN;
02865             if (h2o <= 0)
02866                 continue;
02867
02868             /* Find lifted condensation level (LCL)... */
02869             ptop = P(20.);
02870             pbot = met->ps[ix][iy];
02871             do {
02872                 met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
02873                 t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
02874                 if (RH(met->plcl[ix][iy], t, h2o) > 100.)
02875                     ptop = met->plcl[ix][iy];
02876                 else
02877                     pbot = met->plcl[ix][iy];
02878             } while (pbot - ptop > 0.1);
02879
02880             /* Calculate CIN up to LCL... */
02881             INTPOL_INIT;
02882             double dcape_old, dcape_old, dz, psat, h2o_env, t_env;
02883             double p = met->ps[ix][iy];
02884             met->cape[ix][iy] = met->cin[ix][iy] = 0;
02885             do {
02886                 dz = dz0 * TVIRT(t, h2o);
02887                 p /= pfac;
02888                 t = theta / pow(1000. / p, 0.286);
02889                 intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02890                                     &t_env, ci, cw, 1);
02891                 intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02892                                     &h2o_env, ci, cw, 0);
02893                 dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02894                     TVIRT(t_env, h2o_env) * dz;
02895                 if (dcape < 0)
02896                     met->cin[ix][iy] += fabsf((float) dcape);
02897             } while (p > met->plcl[ix][iy]);
02898
02899             /* Calculate level of free convection (LFC), equilibrium level (EL),
02900              and convective available potential energy (CAPE)... */
02901             dcape = 0;
02902             p = met->plcl[ix][iy];
02903             t = theta / pow(1000. / p, 0.286);
02904             ptop = 0.75 * clim_tropo(met->time, met->lat[iy]);
02905             do {
02906                 dz = dz0 * TVIRT(t, h2o);
02907                 p /= pfac;
02908                 t -= lapse_rate(t, h2o) * dz;
02909                 psat = PSAT(t);
02910                 h2o = psat / (p - (1. - EPS) * psat);
02911                 intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
02912                                     &t_env, ci, cw, 1);
02913                 intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
02914                                     &h2o_env, ci, cw, 0);
02915                 dcape_old = dcape;
02916                 dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
02917                     TVIRT(t_env, h2o_env) * dz;
02918                 if (dcape > 0) {
02919                     met->cape[ix][iy] += (float) dcape;
02920                     if (!isfinite(met->plfc[ix][iy]))
02921                         met->plfc[ix][iy] = (float) p;
02922                 }
02923             } while (p > met->plcl[ix][iy]);

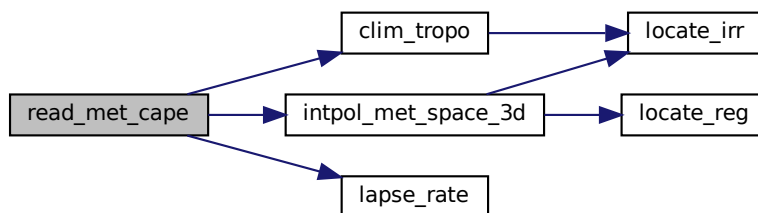
```

```

02922     } else if (dcapc_old > 0)
02923     {
02924         met->pel[ix][iy] = (float) p;
02925         if (dcapc < 0 && !isfinite(met->plfc[ix][iy]))
02926             met->cin[ix][iy] += fabsf((float) dcapc);
02927     } while (p > ptop);
02928
02929     /* Check results... */
02930     if (!isfinite(met->plfc[ix][iy]))
02931         met->cin[ix][iy] = GSL_NAN;
02932 }

```

Here is the call graph for this function:



5.21.3.25 read_met_cloud() void read_met_cloud (
 met_t * met)

Calculate cloud properties.

Definition at line 2936 of file libtrac.c.

```

02937 {
02938
02939     /* Set timer... */
02940     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
02941     LOG(2, "Calculate cloud data...");
02942
02943     /* Loop over columns... */
02944     #pragma omp parallel for default(shared) collapse(2)
02945     for (int ix = 0; ix < met->nx; ix++)
02946         for (int iy = 0; iy < met->ny; iy++) {
02947
02948             /* Init... */
02949             met->pct[ix][iy] = GSL_NAN;
02950             met->pcb[ix][iy] = GSL_NAN;
02951             met->cl[ix][iy] = 0;
02952
02953             /* Loop over pressure levels... */
02954             for (int ip = 0; ip < met->np - 1; ip++) {
02955
02956                 /* Check pressure... */
02957                 if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02958                     continue;
02959
02960                 /* Check ice water and liquid water content... */
02961                 if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0) {
02962
02963                     /* Get cloud top pressure ... */
02964                     met->pct[ix][iy]
02965                         = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
02966
02967                     /* Get cloud bottom pressure ... */
02968                     if (!isfinite(met->pcb[ix][iy]))
02969                         met->pcb[ix][iy]
02970                             = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
02971                 }
02972             }

```

```

02973      /* Get cloud water... */
02974      met->cl[ix][iy] += (float)
02975          (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02976              + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02977              * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02978  }
02979  }
02980  }

```

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

Apply detrending method to temperature and winds.

Definition at line 2984 of file libtrac.c.

```

02986      {
02987
02988      met_t *help;
02989
02990      /* Check parameters... */
02991      if (ctl->met_detrend <= 0)
02992          return;
02993
02994      /* Set timer... */
02995      SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
02996      LOG(2, "Detrend meteo data...");
02997
02998      /* Allocate... */
02999      ALLOC(help, met_t, 1);
03000
03001      /* Calculate standard deviation... */
03002      double sigma = ctl->met_detrend / 2.355;
03003      double tssq = 2. * SQR(sigma);
03004
03005      /* Calculate box size in latitude... */
03006      int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03007      sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03008
03009      /* Calculate background... */
03010      #pragma omp parallel for default(shared) collapse(2)
03011      for (int ix = 0; ix < met->nx; ix++) {
03012          for (int iy = 0; iy < met->ny; iy++) {
03013
03014              /* Calculate Cartesian coordinates... */
03015              double x0[3];
03016              geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03017
03018              /* Calculate box size in longitude... */
03019              int sx =
03020                  (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03021                      fabs(met->lon[1] - met->lon[0]));
03022              sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03023
03024              /* Init... */
03025              float wsum = 0;
03026              for (int ip = 0; ip < met->np; ip++) {
03027                  help->t[ix][iy][ip] = 0;
03028                  help->u[ix][iy][ip] = 0;
03029                  help->v[ix][iy][ip] = 0;
03030                  help->w[ix][iy][ip] = 0;
03031              }
03032
03033              /* Loop over neighboring grid points... */
03034              for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03035                  int ix3 = ix2;
03036                  if (ix3 < 0)
03037                      ix3 += met->nx;
03038                  else if (ix3 >= met->nx)
03039                      ix3 -= met->nx;
03040                  for (int iy2 = GSL_MAX(iy - sy, 0);
03041                      iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03042
03043                      /* Calculate Cartesian coordinates... */
03044                      double x1[3];
03045                      geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03046
03047                      /* Calculate weighting factor... */

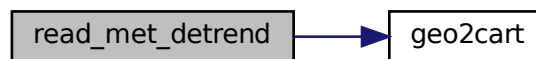
```

```

03048         float w = (float) exp(-DIST2(x0, x1) / tssq);
03049
03050         /* Add data... */
03051         wsum += w;
03052         for (int ip = 0; ip < met->np; ip++) {
03053             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03054             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03055             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03056             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03057         }
03058     }
03059 }
03060
03061     /* Normalize... */
03062     for (int ip = 0; ip < met->np; ip++) {
03063         help->t[ix][iy][ip] /= wsum;
03064         help->u[ix][iy][ip] /= wsum;
03065         help->v[ix][iy][ip] /= wsum;
03066         help->w[ix][iy][ip] /= wsum;
03067     }
03068 }
03069 }
03070
03071     /* Subtract background... */
03072 #pragma omp parallel for default(shared) collapse(3)
03073     for (int ix = 0; ix < met->nx; ix++)
03074         for (int iy = 0; iy < met->ny; iy++)
03075             for (int ip = 0; ip < met->np; ip++) {
03076                 met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03077                 met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03078                 met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03079                 met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03080             }
03081
03082     /* Free... */
03083     free(help);
03084 }

```

Here is the call graph for this function:



5.21.3.27 read_met_extrapolate() void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 3088 of file libtrac.c.

```

03089     {
03090
03091         /* Set timer... */
03092         SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03093         LOG(2, "Extrapolate meteo data...");
03094
03095         /* Loop over columns... */
03096 #pragma omp parallel for default(shared) collapse(2)
03097         for (int ix = 0; ix < met->nx; ix++)
03098             for (int iy = 0; iy < met->ny; iy++) {
03099
03100                 /* Find lowest valid data point... */
03101                 int ip0;
03102                 for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03103                     if (!isfinite(met->t[ix][iy][ip0]))

```

```

03104         || !isfinite(met->u[ix][iy][ip0])
03105         || !isfinite(met->v[ix][iy][ip0])
03106         || !isfinite(met->w[ix][iy][ip0]))
03107     break;
03108
03109     /* Extrapolate... */
03110     for (int ip = ip0; ip >= 0; ip--) {
03111         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03112         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03113         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03114         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03115         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03116         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03117         met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03118         met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03119     }
03120 }
03121 }

```

5.21.3.28 read_met_geopot() void read_met_geopot (

```

    ctl_t * ctl,
    met_t * met )

```

Calculate geopotential heights.

Definition at line 3125 of file libtrac.c.

```

03127     {
03128
03129         static float help[EP][EX][EY];
03130
03131         double logp[EP];
03132
03133         int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03134
03135         /* Set timer... */
03136         SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03137         LOG(2, "Calculate geopotential heights...");
03138
03139         /* Calculate log pressure... */
03140         #pragma omp parallel for default(shared)
03141         for (int ip = 0; ip < met->np; ip++)
03142             logp[ip] = log(met->p[ip]);
03143
03144         /* Apply hydrostatic equation to calculate geopotential heights... */
03145         #pragma omp parallel for default(shared) collapse(2)
03146         for (int ix = 0; ix < met->nx; ix++)
03147             for (int iy = 0; iy < met->ny; iy++) {
03148
03149                 /* Get surface height and pressure... */
03150                 double zs = met->zs[ix][iy];
03151                 double lnps = log(met->ps[ix][iy]);
03152
03153                 /* Get temperature and water vapor vmr at the surface... */
03154                 int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03155                 double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03156                               met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03157                 double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03158                               met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03159
03160                 /* Upper part of profile... */
03161                 met->z[ix][iy][ip0 + 1]
03162                     = (float) (zs +
03163                               ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03164                                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03165                 for (int ip = ip0 + 2; ip < met->np; ip++)
03166                     met->z[ix][iy][ip]
03167                         = (float) (met->z[ix][iy][ip - 1] +
03168                                   ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03169                                         met->h2o[ix][iy][ip - 1], logp[ip],
03170                                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03171
03172                 /* Lower part of profile... */
03173                 met->z[ix][iy][ip0]
03174                     = (float) (zs +
03175                               ZDIFF(lnps, ts, h2os, logp[ip0],
03176                                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03177                 for (int ip = ip0 - 1; ip >= 0; ip--)
03178                     met->z[ix][iy][ip]

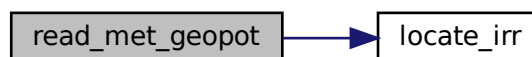
```

```

03179         = (float) (met->z[ix][iy][ip + 1] +
03180                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03181                          met->h2o[ix][iy][ip + 1], logp[ip],
03182                          met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03183     }
03184
03185     /* Check control parameters... */
03186     if (dx == 0 || dy == 0)
03187         return;
03188
03189     /* Default smoothing parameters... */
03190     if (dx < 0 || dy < 0) {
03191         if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03192             dx = 3;
03193             dy = 2;
03194         } else {
03195             dx = 6;
03196             dy = 4;
03197         }
03198     }
03199
03200     /* Calculate weights for smoothing... */
03201     float ws[dx + 1][dy + 1];
03202     #pragma omp parallel for default(shared) collapse(2)
03203     for (int ix = 0; ix <= dx; ix++)
03204         for (int iy = 0; iy < dy; iy++)
03205             ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03206                 * (1.0f - (float) iy / (float) dy);
03207
03208     /* Copy data... */
03209     #pragma omp parallel for default(shared) collapse(3)
03210     for (int ix = 0; ix < met->nx; ix++)
03211         for (int iy = 0; iy < met->ny; iy++)
03212             for (int ip = 0; ip < met->np; ip++)
03213                 help[ip][ix][iy] = met->z[ix][iy][ip];
03214
03215     /* Horizontal smoothing... */
03216     #pragma omp parallel for default(shared) collapse(3)
03217     for (int ip = 0; ip < met->np; ip++)
03218         for (int ix = 0; ix < met->nx; ix++)
03219             for (int iy = 0; iy < met->ny; iy++) {
03220                 float res = 0, wsum = 0;
03221                 int iy0 = GSL_MAX(iy - dy + 1, 0);
03222                 int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03223                 for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03224                     int ix3 = ix2;
03225                     if (ix3 < 0)
03226                         ix3 += met->nx;
03227                     else if (ix3 >= met->nx)
03228                         ix3 -= met->nx;
03229                     for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03230                         if (isfinite(help[ip][ix3][iy2])) {
03231                             float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03232                             res += w * help[ip][ix3][iy2];
03233                             wsum += w;
03234                         }
03235                 }
03236                 if (wsum > 0)
03237                     met->z[ix][iy][ip] = res / wsum;
03238                 else
03239                     met->z[ix][iy][ip] = GSL_NAN;
03240             }
03241 }

```

Here is the call graph for this function:



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

Read coordinates of meteorological data.

Definition at line 3245 of file libtrac.c.

```

03249     {
03250
03251     char levname[LEN], tstr[10];
03252
03253     double rtime, r2;
03254
03255     int dimid, varid, year2, mon2, day2, hour2, min2, sec2;
03256
03257     size_t np, nx, ny;
03258
03259     /* Set timer... */
03260     SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03261     LOG(2, "Read meteo grid information...");
03262
03263     /* Get time from filename... */
03264     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
03265     int year = atoi(tstr);
03266     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
03267     int mon = atoi(tstr);
03268     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
03269     int day = atoi(tstr);
03270     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03271     int hour = atoi(tstr);
03272     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03273
03274     /* Check time... */
03275     if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03276         || day < 1 || day > 31 || hour < 0 || hour > 23)
03277         ERRMSG("Cannot read time from filename!");
03278     jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03279     LOG(2, "Time from filename: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03280         met->time, year2, mon2, day2, hour2, min2);
03281
03282     /* Check time information... */
03283     if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03284         NC(nc_get_var_double(ncid, varid, &rtime));
03285         if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
03286             WARN("Time information in meteo file does not match filename!");
03287     } else
03288         WARN("Time information in meteo file is missing!");
03289
03290     /* Get grid dimensions... */
03291     NC(nc_inq_dimid(ncid, "lon", &dimid));
03292     NC(nc_inq_dimlen(ncid, dimid, &nx));
03293     LOG(2, "Number of longitudes: %zu", nx);
03294     if (nx < 2 || nx > EX)
03295         ERRMSG("Number of longitudes out of range!");
03296
03297     NC(nc_inq_dimid(ncid, "lat", &dimid));
03298     NC(nc_inq_dimlen(ncid, dimid, &ny));
03299     LOG(2, "Number of latitudes: %zu", ny);
03300     if (ny < 2 || ny > EY)
03301         ERRMSG("Number of latitudes out of range!");
03302
03303     sprintf(levname, "lev");
03304     NC(nc_inq_dimid(ncid, levname, &dimid));
03305     NC(nc_inq_dimlen(ncid, dimid, &np));
03306     if (np == 1) {
03307         sprintf(levname, "lev_2");
03308         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03309             sprintf(levname, "plev");
03310             nc_inq_dimid(ncid, levname, &dimid);
03311         }
03312         NC(nc_inq_dimlen(ncid, dimid, &np));
03313     }
03314     LOG(2, "Number of levels: %zu", np);
03315     if (np < 2 || np > EP)
03316         ERRMSG("Number of levels out of range!");
03317
03318     /* Store dimensions... */
03319     met->np = (int) np;
03320     met->nx = (int) nx;
03321     met->ny = (int) ny;
03322

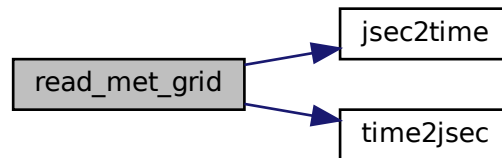
```

```

03323  /* Read longitudes and latitudes... */
03324  NC(nc_inq_varid(ncid, "lon", &varid));
03325  NC(nc_get_var_double(ncid, varid, met->lon));
03326  LOG(2, "Longitudes: %g, %g ... %g deg",
03327       met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03328  NC(nc_inq_varid(ncid, "lat", &varid));
03329  NC(nc_get_var_double(ncid, varid, met->lat));
03330  LOG(2, "Latitudes: %g, %g ... %g deg",
03331       met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03332
03333  /* Read pressure levels... */
03334  if (ctl->met_np <= 0) {
03335      NC(nc_inq_varid(ncid, levname, &varid));
03336      NC(nc_get_var_double(ncid, varid, met->p));
03337      for (int ip = 0; ip < met->np; ip++)
03338          met->p[ip] /= 100.;
03339      LOG(2, "Altitude levels: %g, %g ... %g km",
03340          Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03341      LOG(2, "Pressure levels: %g, %g ... %g hPa",
03342          met->p[0], met->p[1], met->p[met->np - 1]);
03343  }
03344 }

```

Here is the call graph for this function:



5.21.330 read_met_help_3d() int read_met_help_3d (

```

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

```

Read and convert 3D variable from meteorological data file.

Definition at line 3348 of file `libtrac.c`.

```

03355  {
03356
03357      char varsel[LEN];
03358
03359      float offset, scalfac;
03360
03361      int varid;
03362
03363      /* Check if variable exists... */
03364      if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03365          if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03366              WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03367              return 0;
03368          } else {
03369              sprintf(varsel, "%s", varname2);
03370          } else

```

```

03371     sprintf(varsel, "%s", varname);
03372
03373     /* Read packed data... */
03374     if (nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03375         && nc_get_att_float(ncid, varid, "scale_factor",
03376                             &scalfac) == NC_NOERR) {
03377
03378         /* Allocate... */
03379         short *help;
03380         ALLOC(help, short,
03381              EX * EY * EP);
03382
03383         /* Read fill value and missing value... */
03384         short fillval, missval;
03385         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03386             fillval = 0;
03387         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03388             missval = 0;
03389
03390         /* Write info... */
03391         LOG(2, "Read 3-D variable: %s "
03392            "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03393            varsel, fillval, missval, scalfac, offset);
03394
03395         /* Read data... */
03396         NC(nc_get_var_short(ncid, varid, help));
03397
03398         /* Copy and check data... */
03399         #pragma omp parallel for default(shared) num_threads(12)
03400         for (int ix = 0; ix < met->nx; ix++)
03401             for (int iy = 0; iy < met->ny; iy++)
03402                 for (int ip = 0; ip < met->np; ip++) {
03403                     if (init)
03404                         dest[ix][iy][ip] = 0;
03405                     short aux = help[(ip * met->ny + iy) * met->nx + ix];
03406                     if ((fillval == 0 || aux != fillval)
03407                         && (missval == 0 || aux != missval)
03408                         && fabsf(aux * scalfac + offset) < 1e14f)
03409                         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
03410                     else
03411                         dest[ix][iy][ip] = GSL_NAN;
03412                 }
03413
03414         /* Free... */
03415         free(help);
03416     }
03417
03418     /* Unpacked data... */
03419     else {
03420
03421         /* Allocate... */
03422         float *help;
03423         ALLOC(help, float,
03424              EX * EY * EP);
03425
03426         /* Read fill value and missing value... */
03427         float fillval, missval;
03428         if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03429             fillval = 0;
03430         if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03431             missval = 0;
03432
03433         /* Write info... */
03434         LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
03435            varsel, fillval, missval);
03436
03437         /* Read data... */
03438         NC(nc_get_var_float(ncid, varid, help));
03439
03440         /* Copy and check data... */
03441         #pragma omp parallel for default(shared) num_threads(12)
03442         for (int ix = 0; ix < met->nx; ix++)
03443             for (int iy = 0; iy < met->ny; iy++)
03444                 for (int ip = 0; ip < met->np; ip++) {
03445                     if (init)
03446                         dest[ix][iy][ip] = 0;
03447                     float aux = help[(ip * met->ny + iy) * met->nx + ix];
03448                     if ((fillval == 0 || aux != fillval)
03449                         && (missval == 0 || aux != missval)
03450                         && fabsf(aux) < 1e14f)
03451                         dest[ix][iy][ip] += scl * aux;
03452                     else
03453                         dest[ix][iy][ip] = GSL_NAN;
03454                 }
03455
03456         /* Free... */
03457         free(help);

```

```

03458     }
03459
03460     /* Return... */
03461     return 1;
03462 }

```

5.21.3.31 read_met_help_2d() int read_met_help_2d (

```

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

```

Read and convert 2D variable from meteorological data file.

Definition at line 3466 of file libtrac.c.

```

03473     {
03474
03475     char varsel[LEN];
03476
03477     float offset, scalfac;
03478
03479     int varid;
03480
03481     /* Check if variable exists... */
03482     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03483         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03484             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03485             return 0;
03486         } else {
03487             sprintf(varsel, "%s", varname2);
03488         } else
03489             sprintf(varsel, "%s", varname);
03490
03491     /* Read packed data... */
03492     if (nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03493         && nc_get_att_float(ncid, varid, "scale_factor",
03494                             &scalfac) == NC_NOERR) {
03495
03496         /* Write info... */
03497         LOG(2, "Packed: scale_factor= %g / add_offset= %g", scalfac, offset);
03498
03499         /* Allocate... */
03500         short *help;
03501         ALLOC(help, short,
03502              EX * EY * EP);
03503
03504         /* Read fill value and missing value... */
03505         short fillval, missval;
03506         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03507             fillval = 0;
03508         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03509             missval = 0;
03510
03511         /* Write info... */
03512         LOG(2, "Read 2-D variable: %s"
03513              " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03514              varsel, fillval, missval, scalfac, offset);
03515
03516         /* Read data... */
03517         NC(nc_get_var_short(ncid, varid, help));
03518
03519         /* Copy and check data... */
03520 #pragma omp parallel for default(shared) num_threads(12)
03521         for (int ix = 0; ix < met->nx; ix++)
03522             for (int iy = 0; iy < met->ny; iy++) {
03523                 if (init)
03524                     dest[ix][iy] = 0;
03525                 short aux = help[iy * met->nx + ix];
03526                 if ((fillval == 0 || aux != fillval)
03527                     && (missval == 0 || aux != missval)
03528                     && fabsf(aux * scalfac + offset) < 1e14f)
03529                     dest[ix][iy] += scl * (aux * scalfac + offset);
03530                 else

```

```

03531         dest[ix][iy] = GSL_NAN;
03532     }
03533
03534     /* Free... */
03535     free(help);
03536 }
03537
03538 /* Unpacked data... */
03539 else {
03540
03541     /* Allocate... */
03542     float *help;
03543     ALLOC(help, float,
03544           EX * EY);
03545
03546     /* Read fill value and missing value... */
03547     float fillval, missval;
03548     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03549         fillval = 0;
03550     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03551         missval = 0;
03552
03553     /* Write info... */
03554     LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03555         varsel, fillval, missval);
03556
03557     /* Read data... */
03558     NC(nc_get_var_float(ncid, varid, help));
03559
03560     /* Copy and check data... */
03561 #pragma omp parallel for default(shared) num_threads(12)
03562     for (int ix = 0; ix < met->nx; ix++)
03563         for (int iy = 0; iy < met->ny; iy++) {
03564             if (init)
03565                 dest[ix][iy] = 0;
03566             float aux = help[iy * met->nx + ix];
03567             if ((fillval == 0 || aux != fillval)
03568                 && (missval == 0 || aux != missval)
03569                 && fabsf(aux) < 1e14f)
03570                 dest[ix][iy] += scl * aux;
03571             else
03572                 dest[ix][iy] = GSL_NAN;
03573         }
03574
03575     /* Free... */
03576     free(help);
03577 }
03578
03579 /* Return... */
03580 return 1;
03581 }

```

5.21.3.32 read_met_levels() void read_met_levels (

```

    int ncid,
    ctl_t * ctl,
    met_t * met )

```

Read meteorological data on vertical levels.

Definition at line 3585 of file libtrac.c.

```

03588     {
03589
03590     /* Set timer... */
03591     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03592     LOG(2, "Read level data...");
03593
03594     /* Read meteorological data... */
03595     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0, 1))
03596         ERRMSG("Cannot read temperature!");
03597     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0, 1))
03598         ERRMSG("Cannot read zonal wind!");
03599     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0, 1))
03600         ERRMSG("Cannot read meridional wind!");
03601     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f, 1))
03602         WARN("Cannot read vertical velocity!");
03603     if (!read_met_help_3d
03604         (ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O), 1))
03605         WARN("Cannot read specific humidity!");

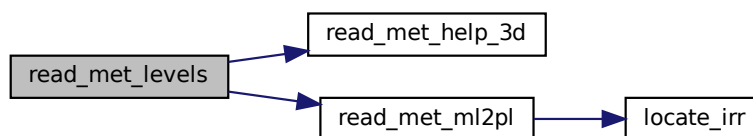
```

```

03606 if (!read_met_help_3d
03607     (ncid, "o3", "O3", met, met->o3, (float) (MA / MO3), 1))
03608     WARN("Cannot read ozone data!");
03609 if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03610     if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0, 1))
03611         WARN("Cannot read cloud liquid water content!");
03612     if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0, 1))
03613         WARN("Cannot read cloud ice water content!");
03614 }
03615 if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03616     if (!read_met_help_3d
03617         (ncid, "crwc", "CRWC", met, met->lwc, 1.0, ctl->met_cloud == 2))
03618         WARN("Cannot read cloud rain water content!");
03619     if (!read_met_help_3d
03620         (ncid, "cswc", "CSWC", met, met->iwc, 1.0, ctl->met_cloud == 2))
03621         WARN("Cannot read cloud snow water content!");
03622 }
03623
03624 /* Transfer from model levels to pressure levels... */
03625 if (ctl->met_np > 0) {
03626
03627     /* Read pressure on model levels... */
03628     if (!read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f, 1))
03629         ERRMSG("Cannot read pressure on model levels!");
03630
03631     /* Vertical interpolation from model to pressure levels... */
03632     read_met_ml2pl(ctl, met, met->t);
03633     read_met_ml2pl(ctl, met, met->u);
03634     read_met_ml2pl(ctl, met, met->v);
03635     read_met_ml2pl(ctl, met, met->w);
03636     read_met_ml2pl(ctl, met, met->h2o);
03637     read_met_ml2pl(ctl, met, met->o3);
03638     read_met_ml2pl(ctl, met, met->lwc);
03639     read_met_ml2pl(ctl, met, met->iwc);
03640
03641     /* Set new pressure levels... */
03642     met->np = ctl->met_np;
03643     for (int ip = 0; ip < met->np; ip++)
03644         met->p[ip] = ctl->met_p[ip];
03645 }
03646
03647 /* Check ordering of pressure levels... */
03648 for (int ip = 1; ip < met->np; ip++)
03649     if (met->p[ip - 1] < met->p[ip])
03650         ERRMSG("Pressure levels must be descending!");
03651 }

```

Here is the call graph for this function:



5.21.3.33 read_met_ml2pl() void read_met_ml2pl (

```

    ctl_t * ctl,
    met_t * met,
    float var[EX][EY][EP] )

```

Convert meteorological data from model levels to pressure levels.

Definition at line 3655 of file libtrac.c.

```

03658 {

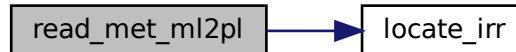
```

```

03659
03660 double aux[EP], p[EP];
03661
03662 /* Set timer... */
03663 SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03664 LOG(2, "Interpolate meteo data to pressure levels...");
03665
03666 /* Loop over columns... */
03667 #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03668 for (int ix = 0; ix < met->nx; ix++)
03669     for (int iy = 0; iy < met->ny; iy++) {
03670
03671         /* Copy pressure profile... */
03672         for (int ip = 0; ip < met->np; ip++)
03673             p[ip] = met->pl[ix][iy][ip];
03674
03675         /* Interpolate... */
03676         for (int ip = 0; ip < ctl->met_np; ip++) {
03677             double pt = ctl->met_p[ip];
03678             if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03679                 pt = p[0];
03680             else if ((pt > p[met->np - 1] && p[1] > p[0])
03681                     || (pt < p[met->np - 1] && p[1] < p[0]))
03682                 pt = p[met->np - 1];
03683             int ip2 = locate_irr(p, met->np, pt);
03684             aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03685                          p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03686         }
03687
03688         /* Copy data... */
03689         for (int ip = 0; ip < ctl->met_np; ip++)
03690             var[ix][iy][ip] = (float) aux[ip];
03691     }
03692 }

```

Here is the call graph for this function:



5.21.3.34 read_met_pbl() void read_met_pbl (
 met_t * met)

Calculate pressure of the boundary layer.

Definition at line 3696 of file libtrac.c.

```

03697 {
03698
03699 /* Set timer... */
03700 SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
03701 LOG(2, "Calculate planetary boundary layer...");
03702
03703 /* Parameters used to estimate the height of the PBL
03704    (e.g., Vogelesang and Holtslag, 1996; Seidel et al., 2012)... */
03705 const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
03706
03707 /* Loop over grid points... */
03708 #pragma omp parallel for default(shared) collapse(2)
03709 for (int ix = 0; ix < met->nx; ix++)
03710     for (int iy = 0; iy < met->ny; iy++) {
03711
03712         /* Set bottom level of PBL... */
03713         double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
03714

```

```

03715      /* Find lowest level near the bottom... */
03716      int ip;
03717      for (ip = 1; ip < met->np; ip++)
03718          if (met->p[ip] < pbl_bot)
03719              break;
03720
03721      /* Get near surface data... */
03722      double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
03723                     met->p[ip], met->z[ix][iy][ip], pbl_bot);
03724      double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
03725                     met->p[ip], met->t[ix][iy][ip], pbl_bot);
03726      double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
03727                     met->p[ip], met->u[ix][iy][ip], pbl_bot);
03728      double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
03729                     met->p[ip], met->v[ix][iy][ip], pbl_bot);
03730      double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
03731                       met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
03732      double tvs = THETA_VIRT(pbl_bot, ts, h2os);
03733
03734      /* Init... */
03735      double rib, rib_old = 0;
03736
03737      /* Loop over levels... */
03738      for (; ip < met->np; ip++) {
03739
03740          /* Get squared horizontal wind speed... */
03741          double vh2
03742              = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
03743          vh2 = GSL_MAX(vh2, SQR(umin));
03744
03745          /* Calculate bulk Richardson number... */
03746          rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
03747                * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
03748                             met->h2o[ix][iy][ip]) - tvs) / vh2;
03749
03750          /* Check for critical value... */
03751          if (rib >= rib_crit) {
03752              met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
03753                                             rib, met->p[ip], rib_crit));
03754              if (met->pbl[ix][iy] > pbl_bot)
03755                  met->pbl[ix][iy] = (float) pbl_bot;
03756              break;
03757          }
03758
03759          /* Save Richardson number... */
03760          rib_old = rib;
03761      }
03762  }
03763 }

```

5.21.3.35 read_met_periodic() void read_met_periodic (
 met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 3767 of file libtrac.c.

```

03768      {
03769
03770          /* Set timer... */
03771          SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
03772          LOG(2, "Apply periodic boundary conditions...");
03773
03774          /* Check longitudes... */
03775          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
03776                  + met->lon[1] - met->lon[0] - 360) < 0.01))
03777              return;
03778
03779          /* Increase longitude counter... */
03780          if ((++met->nx) > EX)
03781              ERRMSG("Cannot create periodic boundary conditions!");
03782
03783          /* Set longitude... */
03784          met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
03785
03786          /* Loop over latitudes and pressure levels... */
03787          #pragma omp parallel for default(shared)
03788          for (int iy = 0; iy < met->ny; iy++) {
03789              met->ps[met->nx - 1][iy] = met->ps[0][iy];
03790              met->zs[met->nx - 1][iy] = met->zs[0][iy];

```



```

03791     met->ts[met->nx - 1][iy] = met->ts[0][iy];
03792     met->us[met->nx - 1][iy] = met->us[0][iy];
03793     met->vs[met->nx - 1][iy] = met->vs[0][iy];
03794     for (int ip = 0; ip < met->np; ip++) {
03795         met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
03796         met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
03797         met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
03798         met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
03799         met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
03800         met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
03801         met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
03802         met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
03803     }
03804 }
03805 }

```

5.21.3.36 read_met_pv() void read_met_pv (
 met_t * met)

Calculate potential vorticity.

Definition at line 3809 of file libtrac.c.

```

03810     {
03811
03812         double pows[EP];
03813
03814         /* Set timer... */
03815         SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
03816         LOG(2, "Calculate potential vorticity...");
03817
03818         /* Set powers... */
03819         #pragma omp parallel for default(shared)
03820         for (int ip = 0; ip < met->np; ip++)
03821             pows[ip] = pow(1000. / met->p[ip], 0.286);
03822
03823         /* Loop over grid points... */
03824         #pragma omp parallel for default(shared)
03825         for (int ix = 0; ix < met->nx; ix++) {
03826
03827             /* Set indices... */
03828             int ix0 = GSL_MAX(ix - 1, 0);
03829             int ix1 = GSL_MIN(ix + 1, met->nx - 1);
03830
03831             /* Loop over grid points... */
03832             for (int iy = 0; iy < met->ny; iy++) {
03833
03834                 /* Set indices... */
03835                 int iy0 = GSL_MAX(iy - 1, 0);
03836                 int iy1 = GSL_MIN(iy + 1, met->ny - 1);
03837
03838                 /* Set auxiliary variables... */
03839                 double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
03840                 double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
03841                 double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
03842                 double c0 = cos(met->lat[iy0] / 180. * M_PI);
03843                 double c1 = cos(met->lat[iy1] / 180. * M_PI);
03844                 double cr = cos(latr / 180. * M_PI);
03845                 double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
03846
03847                 /* Loop over grid points... */
03848                 for (int ip = 0; ip < met->np; ip++) {
03849
03850                     /* Get gradients in longitude... */
03851                     double dtdx
03852                         = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
03853                     double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
03854
03855                     /* Get gradients in latitude... */
03856                     double dtdy
03857                         = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03858                     double dudx
03859                         = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03860
03861                     /* Set indices... */
03862                     int ip0 = GSL_MAX(ip - 1, 0);
03863                     int ip1 = GSL_MIN(ip + 1, met->np - 1);
03864
03865                     /* Get gradients in pressure... */
03866                     double dtdp, dudp, dvdp;

```

```

03867     double dp0 = 100. * (met->p[ip] - met->p[ip0]);
03868     double dp1 = 100. * (met->p[ip1] - met->p[ip]);
03869     if (ip != ip0 && ip != ip1) {
03870         double denom = dp0 * dp1 * (dp0 + dp1);
03871         dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03872             - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03873             + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03874             / denom;
03875         dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03876             - dp1 * dp1 * met->u[ix][iy][ip0]
03877             + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03878             / denom;
03879         dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03880             - dp1 * dp1 * met->v[ix][iy][ip0]
03881             + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03882             / denom;
03883     } else {
03884         double denom = dp0 + dp1;
03885         dtdp =
03886             (met->t[ix][iy][ip1] * pows[ip1] -
03887             met->t[ix][iy][ip0] * pows[ip0]) / denom;
03888         dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03889         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03890     }
03891
03892     /* Calculate PV... */
03893     met->pv[ix][iy][ip] = (float)
03894         (1e6 * G0 *
03895          (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03896 }
03897 }
03898 }
03899
03900 /* Fix for polar regions... */
03901 #pragma omp parallel for default(shared)
03902 for (int ix = 0; ix < met->nx; ix++)
03903     for (int ip = 0; ip < met->np; ip++) {
03904         met->pv[ix][0][ip]
03905             = met->pv[ix][1][ip]
03906             = met->pv[ix][2][ip];
03907         met->pv[ix][met->ny - 1][ip]
03908             = met->pv[ix][met->ny - 2][ip]
03909             = met->pv[ix][met->ny - 3][ip];
03910     }
03911 }

```

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

Downsampling of meteorological data.

Definition at line 3915 of file libtrac.c.

```

03917     {
03918
03919     met_t *help;
03920
03921     /* Check parameters... */
03922     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03923         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03924         return;
03925
03926     /* Set timer... */
03927     SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
03928     LOG(2, "Downsampling of meteo data...");
03929
03930     /* Allocate... */
03931     ALLOC(help, met_t, 1);
03932
03933     /* Copy data... */
03934     help->nx = met->nx;
03935     help->ny = met->ny;
03936     help->np = met->np;
03937     memcpy(help->lon, met->lon, sizeof(met->lon));
03938     memcpy(help->lat, met->lat, sizeof(met->lat));
03939     memcpy(help->p, met->p, sizeof(met->p));
03940
03941     /* Smoothing... */

```

```

03942     for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
03943         for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
03944             for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
03945                 help->ps[ix][iy] = 0;
03946                 help->zs[ix][iy] = 0;
03947                 help->ts[ix][iy] = 0;
03948                 help->us[ix][iy] = 0;
03949                 help->vs[ix][iy] = 0;
03950                 help->t[ix][iy][ip] = 0;
03951                 help->u[ix][iy][ip] = 0;
03952                 help->v[ix][iy][ip] = 0;
03953                 help->w[ix][iy][ip] = 0;
03954                 help->h2o[ix][iy][ip] = 0;
03955                 help->o3[ix][iy][ip] = 0;
03956                 help->lwc[ix][iy][ip] = 0;
03957                 help->iwc[ix][iy][ip] = 0;
03958                 float wsum = 0;
03959                 for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
03960                     ix2++) {
03961                     int ix3 = ix2;
03962                     if (ix3 < 0)
03963                         ix3 += met->nx;
03964                     else if (ix3 >= met->nx)
03965                         ix3 -= met->nx;
03966
03967                     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03968                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03969                         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03970                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03971                             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
03972                                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
03973                                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
03974                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
03975                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
03976                             help->ts[ix][iy] += w * met->ts[ix3][iy2];
03977                             help->us[ix][iy] += w * met->us[ix3][iy2];
03978                             help->vs[ix][iy] += w * met->vs[ix3][iy2];
03979                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03980                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03981                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03982                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03983                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03984                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03985                             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03986                             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03987                             wsum += w;
03988                         }
03989                 }
03990                 help->ps[ix][iy] /= wsum;
03991                 help->zs[ix][iy] /= wsum;
03992                 help->ts[ix][iy] /= wsum;
03993                 help->us[ix][iy] /= wsum;
03994                 help->vs[ix][iy] /= wsum;
03995                 help->t[ix][iy][ip] /= wsum;
03996                 help->u[ix][iy][ip] /= wsum;
03997                 help->v[ix][iy][ip] /= wsum;
03998                 help->w[ix][iy][ip] /= wsum;
03999                 help->h2o[ix][iy][ip] /= wsum;
04000                 help->o3[ix][iy][ip] /= wsum;
04001                 help->lwc[ix][iy][ip] /= wsum;
04002                 help->iwc[ix][iy][ip] /= wsum;
04003             }
04004         }
04005     }
04006
04007     /* Downsampling... */
04008     met->nx = 0;
04009     for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04010         met->lon[met->nx] = help->lon[ix];
04011         met->ny = 0;
04012         for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04013             met->lat[met->ny] = help->lat[iy];
04014             met->ps[met->nx][met->ny] = help->ps[ix][iy];
04015             met->zs[met->nx][met->ny] = help->zs[ix][iy];
04016             met->ts[met->nx][met->ny] = help->ts[ix][iy];
04017             met->us[met->nx][met->ny] = help->us[ix][iy];
04018             met->vs[met->nx][met->ny] = help->vs[ix][iy];
04019             met->np = 0;
04020             for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04021                 met->p[met->np] = help->p[ip];
04022                 met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04023                 met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04024                 met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04025                 met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04026                 met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04027                 met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04028                 met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];

```

```

04029         met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04030         met->np++;
04031     }
04032     met->ny++;
04033 }
04034     met->nx++;
04035 }
04036
04037 /* Free... */
04038 free(help);
04039 }

```

5.21.3.38 read_met_surface() void read_met_surface (

int ncid,

met_t * met)

Read surface data.

Definition at line 4043 of file libtrac.c.

```

04045     {
04046
04047     /* Set timer... */
04048     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04049     LOG(2, "Read surface data...");
04050
04051     /* Read surface pressure... */
04052     if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0f, 1)) {
04053         if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f, 1)) {
04054             WARN("Cannot not read surface pressure data (use lowest level!)");
04055             for (int ix = 0; ix < met->nx; ix++)
04056                 for (int iy = 0; iy < met->ny; iy++)
04057                     met->ps[ix][iy] = (float) met->p[0];
04058         }
04059     } else
04060         for (int ix = 0; ix < met->nx; ix++)
04061             for (int iy = 0; iy < met->ny; iy++)
04062                 met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04063
04064     /* Read geopotential height at the surface... */
04065     if (!read_met_help_2d
04066         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0)), 1))
04067         if (!read_met_help_2d
04068             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.), 1))
04069             WARN("Cannot read surface geopotential height!");
04070
04071     /* Read temperature at the surface... */
04072     if (!read_met_help_2d(ncid, "t2m", "T2M", met, met->ts, 1.0, 1))
04073         WARN("Cannot read surface temperature!");
04074
04075     /* Read zonal wind at the surface... */
04076     if (!read_met_help_2d(ncid, "u10m", "U10M", met, met->us, 1.0, 1))
04077         WARN("Cannot read surface zonal wind!");
04078
04079     /* Read meridional wind at the surface... */
04080     if (!read_met_help_2d(ncid, "v10m", "V10M", met, met->vs, 1.0, 1))
04081         WARN("Cannot read surface meridional wind!");
04082 }

```

Here is the call graph for this function:



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

Calculate tropopause data.

Definition at line 4086 of file libtrac.c.

```

04088     {
04089
04090         double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04091             th2[200], z[EP], z2[200];
04092
04093         /* Set timer... */
04094         SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04095         LOG(2, "Calculate tropopause...");
04096
04097         /* Get altitude and pressure profiles... */
04098         #pragma omp parallel for default(shared)
04099         for (int iz = 0; iz < met->np; iz++)
04100             z[iz] = Z(met->p[iz]);
04101         #pragma omp parallel for default(shared)
04102         for (int iz = 0; iz <= 190; iz++) {
04103             z2[iz] = 4.5 + 0.1 * iz;
04104             p2[iz] = P(z2[iz]);
04105         }
04106
04107         /* Do not calculate tropopause... */
04108         if (ctl->met_tropo == 0)
04109             #pragma omp parallel for default(shared) collapse(2)
04110             for (int ix = 0; ix < met->nx; ix++)
04111                 for (int iy = 0; iy < met->ny; iy++)
04112                     met->pt[ix][iy] = GSL_NAN;
04113
04114         /* Use tropopause climatology... */
04115         else if (ctl->met_tropo == 1) {
04116             #pragma omp parallel for default(shared) collapse(2)
04117             for (int ix = 0; ix < met->nx; ix++)
04118                 for (int iy = 0; iy < met->ny; iy++)
04119                     met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
04120         }
04121
04122         /* Use cold point... */
04123         else if (ctl->met_tropo == 2) {
04124
04125             /* Loop over grid points... */
04126             #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04127             for (int ix = 0; ix < met->nx; ix++)
04128                 for (int iy = 0; iy < met->ny; iy++) {
04129
04130                 /* Interpolate temperature profile... */
04131                 for (int iz = 0; iz < met->np; iz++)
04132                     t[iz] = met->t[ix][iy][iz];
04133                 spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04134
04135                 /* Find minimum... */
04136                 int iz = (int) gsl_stats_min_index(t2, 1, 171);
04137                 if (iz > 0 && iz < 170)
04138                     met->pt[ix][iy] = (float) p2[iz];
04139                 else
04140                     met->pt[ix][iy] = GSL_NAN;
04141             }
04142         }
04143
04144         /* Use WMO definition... */
04145         else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04146
04147             /* Loop over grid points... */
04148             #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04149             for (int ix = 0; ix < met->nx; ix++)
04150                 for (int iy = 0; iy < met->ny; iy++) {
04151
04152                 /* Interpolate temperature profile... */
04153                 int iz;
04154                 for (iz = 0; iz < met->np; iz++)
04155                     t[iz] = met->t[ix][iy][iz];
04156                 spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04157
04158                 /* Find 1st tropopause... */
04159                 met->pt[ix][iy] = GSL_NAN;
04160                 for (iz = 0; iz <= 170; iz++) {
04161                     int found = 1;
04162                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04163                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04164                             ctl->met_tropo_lapse) {

```

```

04165         found = 0;
04166         break;
04167     }
04168     if (found) {
04169         if (iz > 0 && iz < 170)
04170             met->pt[ix][iy] = (float) p2[iz];
04171         break;
04172     }
04173 }
04174
04175 /* Find 2nd tropopause... */
04176 if (ctl->met_tropo == 4) {
04177     met->pt[ix][iy] = GSL_NAN;
04178     for (; iz <= 170; iz++) {
04179         int found = 1;
04180         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04181             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04182                 ctl->met_tropo_lapse_sep) {
04183                 found = 0;
04184                 break;
04185             }
04186         if (found)
04187             break;
04188     }
04189     for (; iz <= 170; iz++) {
04190         int found = 1;
04191         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04192             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04193                 ctl->met_tropo_lapse) {
04194                 found = 0;
04195                 break;
04196             }
04197         if (found) {
04198             if (iz > 0 && iz < 170)
04199                 met->pt[ix][iy] = (float) p2[iz];
04200             break;
04201         }
04202     }
04203 }
04204 }
04205 }
04206
04207 /* Use dynamical tropopause... */
04208 else if (ctl->met_tropo == 5) {
04209     /* Loop over grid points... */
04210     #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04211     for (int ix = 0; ix < met->nx; ix++)
04212         for (int iy = 0; iy < met->ny; iy++) {
04213
04214             /* Interpolate potential vorticity profile... */
04215             for (int iz = 0; iz < met->np; iz++)
04216                 pv[iz] = met->pv[ix][iy][iz];
04217             spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04218
04219             /* Interpolate potential temperature profile... */
04220             for (int iz = 0; iz < met->np; iz++)
04221                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04222             spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04223
04224             /* Find dynamical tropopause... */
04225             met->pt[ix][iy] = GSL_NAN;
04226             for (int iz = 0; iz <= 170; iz++)
04227                 if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04228                     || th2[iz] >= ctl->met_tropo_theta) {
04229                     if (iz > 0 && iz < 170)
04230                         met->pt[ix][iy] = (float) p2[iz];
04231                     break;
04232                 }
04233         }
04234     }
04235 }
04236
04237 else
04238     ERRMSG("Cannot calculate tropopause!");
04239
04240 /* Interpolate temperature, geopotential height, and water vapor vmr... */
04241 #pragma omp parallel for default(shared) collapse(2)
04242 for (int ix = 0; ix < met->nx; ix++)
04243     for (int iy = 0; iy < met->ny; iy++) {
04244         double h2ot, tt, zt;
04245         INTPOL_INIT;
04246         intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04247                             met->lat[iy], &tt, ci, cw, 1);
04248         intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04249                             met->lat[iy], &zt, ci, cw, 0);
04250         intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04251                             met->lat[iy], &h2ot, ci, cw, 0);

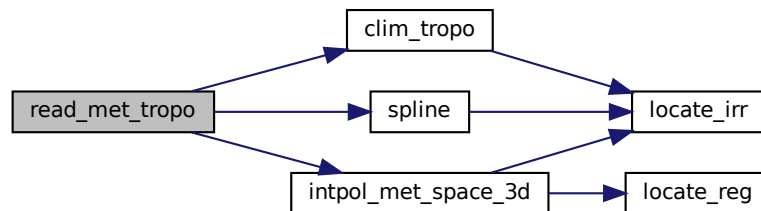
```

```

04252     met->tt[ix][iy] = (float) tt;
04253     met->zr[ix][iy] = (float) zr;
04254     met->h2ot[ix][iy] = (float) h2ot;
04255 }
04256 }

```

Here is the call graph for this function:



5.21.3.40 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 4260 of file libtrac.c.

```

04267     {
04268
04269     FILE *in = NULL;
04270
04271     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
04272           rvarname[LEN], rval[LEN];
04273
04274     int contain = 0, i;
04275
04276     /* Open file... */
04277     if (filename[strlen(filename) - 1] != '-')
04278         if (!(in = fopen(filename, "r")))
04279             ERRMSG("Cannot open file!");
04280
04281     /* Set full variable name... */
04282     if (arridx >= 0) {
04283         sprintf(fullname1, "%s[%d]", varname, arridx);
04284         sprintf(fullname2, "%s[*]", varname);
04285     } else {
04286         sprintf(fullname1, "%s", varname);
04287         sprintf(fullname2, "%s", varname);
04288     }
04289
04290     /* Read data... */
04291     if (in != NULL)
04292         while (fgets(line, LEN, in))
04293             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
04294                 if (strcasecmp(rvarname, fullname1) == 0 ||
04295                     strcasecmp(rvarname, fullname2) == 0) {
04296                     contain = 1;
04297                     break;
04298                 }
04299     for (i = 1; i < argc - 1; i++)

```

```

04300     if (strcasecmp(argv[i], fullname1) == 0 ||
04301         strcasecmp(argv[i], fullname2) == 0) {
04302         sprintf(rval, "%s", argv[i + 1]);
04303         contain = 1;
04304         break;
04305     }
04306
04307     /* Close file... */
04308     if (in != NULL)
04309         fclose(in);
04310
04311     /* Check for missing variables... */
04312     if (!contain) {
04313         if (strlen(defvalue) > 0)
04314             sprintf(rval, "%s", defvalue);
04315         else
04316             ERRMSG("Missing variable %s!\n", fullname1);
04317     }
04318
04319     /* Write info... */
04320     LOG(1, "%s = %s", fullname1, rval);
04321
04322     /* Return values... */
04323     if (value != NULL)
04324         sprintf(value, "%s", rval);
04325     return atof(rval);
04326 }

```

5.21.3.41 sedi() double sedi (
 double p,
 double T,
 double r_p,
 double rho_p)

Calculate sedimentation velocity.

Definition at line 4330 of file libtrac.c.

```

04334     {
04335
04336     double eta, G, K, lambda, rho, v;
04337
04338     /* Convert pressure from hPa to Pa... */
04339     p *= 100.;
04340
04341     /* Convert particle radius from microns to m... */
04342     r_p *= 1e-6;
04343
04344     /* Density of dry air [kg / m^3]... */
04345     rho = p / (RA * T);
04346
04347     /* Dynamic viscosity of air [kg / (m s)]... */
04348     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04349
04350     /* Thermal velocity of an air molecule [m / s]... */
04351     v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04352
04353     /* Mean free path of an air molecule [m]... */
04354     lambda = 2. * eta / (rho * v);
04355
04356     /* Knudsen number for air (dimensionless)... */
04357     K = lambda / r_p;
04358
04359     /* Cunningham slip-flow correction (dimensionless)... */
04360     G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04361
04362     /* Sedimentation velocity [m / s]... */
04363     return 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
04364 }

```


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

Spline interpolation.

Definition at line 4368 of file libtrac.c.

```

04375     {
04376
04377     /* Cubic spline interpolation... */
04378     if (method == 1) {
04379
04380     /* Allocate... */
04381     gsl_interp_accel *acc;
04382     gsl_spline *s;
04383     acc = gsl_interp_accel_alloc();
04384     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04385
04386     /* Interpolate profile... */
04387     gsl_spline_init(s, x, y, (size_t) n);
04388     for (int i = 0; i < n2; i++)
04389         if (x2[i] <= x[0])
04390             y2[i] = y[0];
04391         else if (x2[i] >= x[n - 1])
04392             y2[i] = y[n - 1];
04393         else
04394             y2[i] = gsl_spline_eval(s, x2[i], acc);
04395
04396     /* Free... */
04397     gsl_spline_free(s);
04398     gsl_interp_accel_free(acc);
04399     }
04400
04401     /* Linear interpolation... */
04402     else {
04403     for (int i = 0; i < n2; i++)
04404         if (x2[i] <= x[0])
04405             y2[i] = y[0];
04406         else if (x2[i] >= x[n - 1])
04407             y2[i] = y[n - 1];
04408         else {
04409             int idx = locate_irr(x, n, x2[i]);
04410             y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04411         }
04412     }
04413 }
```

Here is the call graph for this function:



5.21.3.43 stddev() float stddev (
float * data,
int n)

Calculate standard deviation.

Definition at line 4417 of file [libtrac.c](#).

```
04419     {
04420
04421     if (n <= 0)
04422         return 0;
04423
04424     float mean = 0, var = 0;
04425
04426     for (int i = 0; i < n; ++i) {
04427         mean += data[i];
04428         var += SQR(data[i]);
04429     }
04430
04431     return sqrtf(var / (float) n - SQR(mean / (float) n));
04432 }
```

5.21.3.44 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 4436 of file [libtrac.c](#).

```
04444     {
04445
04446     struct tm t0, t1;
04447
04448     t0.tm_year = 100;
04449     t0.tm_mon = 0;
04450     t0.tm_mday = 1;
04451     t0.tm_hour = 0;
04452     t0.tm_min = 0;
04453     t0.tm_sec = 0;
04454
04455     t1.tm_year = year - 1900;
04456     t1.tm_mon = mon - 1;
04457     t1.tm_mday = day;
04458     t1.tm_hour = hour;
04459     t1.tm_min = min;
04460     t1.tm_sec = sec;
04461
04462     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04463 }
```

5.21.3.45 timer() void timer (

```
    const char * name,
    const char * group,
    int output )
```

Measure wall-clock time.

Definition at line 4467 of file [libtrac.c](#).

```
04470     {
04471
04472     static char names[NTIMER][100], groups[NTIMER][100];
04473
04474     static double rt_name[NTIMER], rt_group[NTIMER], t0, t1;
```

```

04475
04476 static int iname = -1, igrp = -1, nname, ngrp;
04477
04478 /* Get time... */
04479 t1 = omp_get_wtime();
04480
04481 /* Add elapsed time to current timers... */
04482 if (iname >= 0)
04483     rt_name[iname] += t1 - t0;
04484 if (igrp >= 0)
04485     rt_group[igrp] += t1 - t0;
04486
04487 /* Report timers... */
04488 if (output) {
04489     for (int i = 0; i < nname; i++)
04490         LOG(1, "TIMER_%s = %.3f s", names[i], rt_name[i]);
04491     for (int i = 0; i < ngrp; i++)
04492         LOG(1, "TIMER_%s = %.3f s", groups[i], rt_group[i]);
04493     double total = 0.0;
04494     for (int i = 0; i < nname; i++)
04495         total += rt_name[i];
04496     LOG(1, "TIMER_TOTAL = %.3f s", total);
04497 }
04498
04499 /* Identify IDs of next timer... */
04500 for (iname = 0; iname < nname; iname++)
04501     if (strcasecmp(name, names[iname]) == 0)
04502         break;
04503 for (igrp = 0; igrp < ngrp; igrp++)
04504     if (strcasecmp(group, groups[igrp]) == 0)
04505         break;
04506
04507 /* Check whether this is a new timer... */
04508 if (iname >= nname) {
04509     sprintf(names[iname], "%s", name);
04510     if ((++nname) > NTIMER)
04511         ERRMSG("Too many timers!");
04512 }
04513
04514 /* Check whether this is a new group... */
04515 if (igrp >= ngrp) {
04516     sprintf(groups[igrp], "%s", group);
04517     if ((++ngrp) > NTIMER)
04518         ERRMSG("Too many groups!");
04519 }
04520
04521 /* Save starting time... */
04522 t0 = t1;
04523 }

```

5.21.3.46 tropo_weight() double tropo_weight (
double t,
double lat,
double p)

Get weighting factor based on tropopause distance.

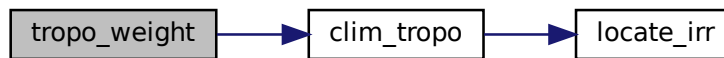
Definition at line 4527 of file libtrac.c.

```

04530 {
04531
04532 /* Get tropopause pressure... */
04533 double pt = clim_tropo(t, lat);
04534
04535 /* Get pressure range... */
04536 double p1 = pt * 0.866877899;
04537 double p0 = pt / 0.866877899;
04538
04539 /* Get weighting factor... */
04540 if (p > p0)
04541     return 1;
04542 else if (p < p1)
04543     return 0;
04544 else
04545     return LIN(p0, 1.0, p1, 0.0, p);
04546 }

```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 4550 of file libtrac.c.

```

04554     {
04555
04556     FILE *in, *out;
04557
04558     char line[LEN];
04559
04560     double r, t0, t1;
04561
04562     int year, mon, day, hour, min, sec;
04563
04564     /* Set timer... */
04565     SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
04566
04567     /* Set time interval for output... */
04568     t0 = t - 0.5 * ctl->dt_mod;
04569     t1 = t + 0.5 * ctl->dt_mod;
04570
04571     /* Write info... */
04572     LOG(1, "Write atmospheric data: %s", filename);
04573
04574     /* Write ASCII data... */
04575     if (ctl->atm_type == 0) {
04576
04577         /* Check if gnuplot output is requested... */
04578         if (ctl->atm_gpfile[0] != '-') {
04579
04580             /* Create gnuplot pipe... */
04581             if (!(out = popen("gnuplot", "w")))
04582                 ERRMSG("Cannot create pipe to gnuplot!");
04583
04584             /* Set plot filename... */
04585             fprintf(out, "set out \"%s.png\"\n", filename);
04586
04587             /* Set time string... */
04588             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04589             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04590                 year, mon, day, hour, min);
04591
04592             /* Dump gnuplot file to pipe... */
04593             if (!(in = fopen(ctl->atm_gpfile, "r")))
04594                 ERRMSG("Cannot open file!");
04595             while (fgets(line, LEN, in))
04596                 fprintf(out, "%s", line);
04597             fclose(in);
04598         }
04599
04600     else {
04601
04602         /* Create file... */
04603         if (!(out = fopen(filename, "w")))
  
```

```

04604         ERRMSG("Cannot create file!");
04605     }
04606
04607     /* Write header... */
04608     fprintf(out,
04609         "# $1 = time [s]\n"
04610         "# $2 = altitude [km]\n"
04611         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04612     for (int iq = 0; iq < ctl->nq; iq++)
04613         fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
04614             ctl->qnt_unit[iq]);
04615     fprintf(out, "\n");
04616
04617     /* Write data... */
04618     for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
04619
04620         /* Check time... */
04621         if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
04622             continue;
04623
04624         /* Write output... */
04625         fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
04626             atm->lon[ip], atm->lat[ip]);
04627         for (int iq = 0; iq < ctl->nq; iq++) {
04628             fprintf(out, " ");
04629             if (ctl->atm_filter == 1
04630                 && (atm->time[ip] < t0 || atm->time[ip] > t1))
04631                 fprintf(out, ctl->qnt_format[iq], GSL_NAN);
04632             else
04633                 fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04634         }
04635         fprintf(out, "\n");
04636     }
04637
04638     /* Close file... */
04639     fclose(out);
04640 }
04641
04642 /* Write binary data... */
04643 else if (ctl->atm_type == 1) {
04644
04645     /* Create file... */
04646     if (!(out = fopen(filename, "w")))
04647         ERRMSG("Cannot create file!");
04648
04649     /* Write data... */
04650     FWRITE(&atm->np, int,
04651         1,
04652         out);
04653     FWRITE(atm->time, double,
04654         (size_t) atm->np,
04655         out);
04656     FWRITE(atm->p, double,
04657         (size_t) atm->np,
04658         out);
04659     FWRITE(atm->lon, double,
04660         (size_t) atm->np,
04661         out);
04662     FWRITE(atm->lat, double,
04663         (size_t) atm->np,
04664         out);
04665     for (int iq = 0; iq < ctl->nq; iq++)
04666         FWRITE(atm->q[iq], double,
04667             (size_t) atm->np,
04668             out);
04669
04670     /* Close file... */
04671     fclose(out);
04672 }
04673
04674 /* Error... */
04675 else
04676     ERRMSG("Atmospheric data type not supported!");
04677
04678 /* Write info... */
04679 double mini, maxi;
04680 LOG(2, "Number of particles: %d", atm->np);
04681 gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
04682 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04683 gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
04684 LOG(2, "Altitude range: %g ... %g km", Z(mini), Z(maxi));
04685 LOG(2, "Pressure range: %g ... %g hPa", mini, maxi);
04686 gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
04687 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04688 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
04689 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04690 for (int iq = 0; iq < ctl->nq; iq++) {

```

```

04691     char msg[LEN];
04692     sprintf(msg, "Quantity %s range: %s ... %s %s",
04693             ctl->qnt_name[iq], ctl->qnt_format[iq],
04694             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
04695     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
04696     LOG(2, msg, mini, maxi);
04697 }
04698 }

```

Here is the call graph for this function:



5.21.3.48 write_csi() void write_csi (

```

    const char * filename,
    ctl_t * ctl,
    atm_t * atm,
    double t )

```

Write CSI data.

Definition at line 4702 of file libtrac.c.

```

04706     {
04707
04708     static FILE *in, *out;
04709
04710     static char line[LEN];
04711
04712     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ], rt, rt_old,
04713             rz, rlon, rlat, robs, t0, t1, area[GY], dlon, dlat, dz, lat,
04714             x[1000000], y[1000000], work[2000000];
04715
04716     static int obscount[GX][GY][GZ], ct, cx, cy, cz, ip, ix, iy, iz, n;
04717
04718     /* Set timer... */
04719     SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
04720
04721     /* Init... */
04722     if (t == ctl->t_start) {
04723
04724         /* Check quantity index for mass... */
04725         if (ctl->qnt_m < 0)
04726             ERRMSG("Need quantity mass!");
04727
04728         /* Open observation data file... */
04729         LOG(1, "Read CSI observation data: %s", ctl->csi_obsfile);
04730         if (!(in = fopen(ctl->csi_obsfile, "r")))
04731             ERRMSG("Cannot open file!");
04732
04733         /* Initialize time for file input... */
04734         rt_old = -1e99;
04735
04736         /* Create new file... */
04737         LOG(1, "Write CSI data: %s", filename);
04738         if (!(out = fopen(filename, "w")))
04739             ERRMSG("Cannot create file!");
04740
04741         /* Write header... */
04742         fprintf(out,
04743             "# $1 = time [s]\n"
04744             "# $2 = number of hits (cx)\n"
04745             "# $3 = number of misses (cy)\n"

```

```

04746         "# $4 = number of false alarms (cz)\n"
04747         "# $5 = number of observations (cx + cy)\n"
04748         "# $6 = number of forecasts (cx + cz)\n"
04749         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
04750         "# $8 = probability of detection (POD) [%%]\n"
04751         "# $9 = false alarm rate (FAR) [%%]\n"
04752         "# $10 = critical success index (CSI) [%%]\n");
04753     fprintf(out,
04754         "# $11 = hits associated with random chance\n"
04755         "# $12 = equitable threat score (ETS) [%%]\n"
04756         "# $13 = Pearson linear correlation coefficient\n"
04757         "# $14 = Spearman rank-order correlation coefficient\n"
04758         "# $15 = column density mean error (F - O) [kg/m^2]\n"
04759         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
04760         "# $17 = column density mean absolute error [kg/m^2]\n"
04761         "# $18 = number of data points\n\n");
04762
04763     /* Set grid box size... */
04764     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
04765     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
04766     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
04767
04768     /* Set horizontal coordinates... */
04769     for (iy = 0; iy < ctl->csi_ny; iy++) {
04770         lat = ctl->csi_lat0 + dlat * (iy + 0.5);
04771         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
04772     }
04773 }
04774
04775 /* Set time interval... */
04776 t0 = t - 0.5 * ctl->dt_mod;
04777 t1 = t + 0.5 * ctl->dt_mod;
04778
04779 /* Initialize grid cells... */
04780 #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(3)
04781 for (ix = 0; ix < ctl->csi_nx; ix++)
04782     for (iy = 0; iy < ctl->csi_ny; iy++)
04783         for (iz = 0; iz < ctl->csi_nz; iz++)
04784             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
04785
04786 /* Read observation data... */
04787 while (fgets(line, LEN, in)) {
04788
04789     /* Read data... */
04790     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
04791         5)
04792         continue;
04793
04794     /* Check time... */
04795     if (rt < t0)
04796         continue;
04797     if (rt > t1)
04798         break;
04799     if (rt < rt_old)
04800         ERRMSG("Time must be ascending!");
04801     rt_old = rt;
04802
04803     /* Check observation data... */
04804     if (!isfinite(robs))
04805         continue;
04806
04807     /* Calculate indices... */
04808     ix = (int) ((rlon - ctl->csi_lon0) / dlon);
04809     iy = (int) ((rlat - ctl->csi_lat0) / dlat);
04810     iz = (int) ((rz - ctl->csi_z0) / dz);
04811
04812     /* Check indices... */
04813     if (ix < 0 || ix >= ctl->csi_nx ||
04814         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04815         continue;
04816
04817     /* Get mean observation index... */
04818     obsmean[ix][iy][iz] += robs;
04819     obscount[ix][iy][iz]++;
04820 }
04821
04822 /* Analyze model data... */
04823 for (ip = 0; ip < atm->np; ip++) {
04824
04825     /* Check time... */
04826     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04827         continue;
04828
04829     /* Get indices... */
04830     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
04831     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
04832     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);

```

```

04833
04834 /* Check indices... */
04835 if (ix < 0 || ix >= ctl->csi_nx ||
04836     iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
04837     continue;
04838
04839 /* Get total mass in grid cell... */
04840 modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04841 }
04842
04843 /* Analyze all grid cells... */
04844 for (ix = 0; ix < ctl->csi_nx; ix++)
04845     for (iy = 0; iy < ctl->csi_ny; iy++)
04846         for (iz = 0; iz < ctl->csi_nz; iz++) {
04847
04848             /* Calculate mean observation index... */
04849             if (obscount[ix][iy][iz] > 0)
04850                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
04851
04852             /* Calculate column density... */
04853             if (modmean[ix][iy][iz] > 0)
04854                 modmean[ix][iy][iz] /= (1e6 * area[iy]);
04855
04856             /* Calculate CSI... */
04857             if (obscount[ix][iy][iz] > 0) {
04858                 ct++;
04859                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04860                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04861                     cx++;
04862                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
04863                     modmean[ix][iy][iz] < ctl->csi_modmin)
04864                     cy++;
04865                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
04866                     modmean[ix][iy][iz] >= ctl->csi_modmin)
04867                     cz++;
04868             }
04869
04870             /* Save data for other verification statistics... */
04871             if (obscount[ix][iy][iz] > 0
04872                 && (obsmean[ix][iy][iz] >= ctl->csi_obsmin
04873                     || modmean[ix][iy][iz] >= ctl->csi_modmin)) {
04874                 x[n] = modmean[ix][iy][iz];
04875                 y[n] = obsmean[ix][iy][iz];
04876                 if ((++n) > 1000000)
04877                     ERRMSG("Too many data points to calculate statistics!");
04878             }
04879         }
04880
04881 /* Write output... */
04882 if (fmod(t, ctl->csi_dt_out) == 0) {
04883
04884     /* Calculate verification statistics
04885      (https://www.cawcr.gov.au/projects/verification/) ... */
04886     int nobs = cx + cy;
04887     int nfor = cx + cz;
04888     double bias = (nobs > 0) ? 100. * nfor / nobs : GSL_NAN;
04889     double pod = (nobs > 0) ? (100. * cx) / nobs : GSL_NAN;
04890     double far = (nfor > 0) ? (100. * cz) / nfor : GSL_NAN;
04891     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
04892     double cx_rd = (ct > 0) ? (1. * nobs * nfor) / ct : GSL_NAN;
04893     double ets = (cx + cy + cz - cx_rd > 0) ?
04894         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
04895     double rho_p =
04896         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
04897     double rho_s =
04898         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
04899     for (int i = 0; i < n; i++)
04900         work[i] = x[i] - y[i];
04901     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
04902     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
04903         0.0) : GSL_NAN;
04904     double absdev =
04905         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
04906
04907     /* Write... */
04908     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %d\n",
04909         t, cx, cy, cz, nobs, nfor, bias, pod, far, csi, cx_rd, ets,
04910         rho_p, rho_s, mean, rmse, absdev, n);
04911
04912     /* Set counters to zero... */
04913     n = ct = cx = cy = cz = 0;
04914 }
04915
04916 /* Close file... */
04917 if (t == ctl->t_stop)
04918     fclose(out);
04919 }

```


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

Write ensemble data.

Definition at line 4923 of file libtrac.c.

```

04927     {
04928
04929         static FILE *out;
04930
04931         static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
04932             t0, t1, x[NENS][3], xm[3];
04933
04934         static int ip, iq;
04935
04936         static size_t i, n;
04937
04938         /* Set timer... */
04939         SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
04940
04941         /* Init... */
04942         if (t == ctl->t_start) {
04943
04944             /* Check quantities... */
04945             if (ctl->qnt_ens < 0)
04946                 ERRMSG("Missing ensemble IDs!");
04947
04948             /* Create new file... */
04949             LOG(1, "Write ensemble data: %s", filename);
04950             if (!(out = fopen(filename, "w")))
04951                 ERRMSG("Cannot create file!");
04952
04953             /* Write header... */
04954             fprintf(out,
04955                 "# $1 = time [s]\n"
04956                 "# $2 = altitude [km]\n"
04957                 "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04958             for (iq = 0; iq < ctl->nq; iq++)
04959                 fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
04960                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04961             for (iq = 0; iq < ctl->nq; iq++)
04962                 fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
04963                     ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04964             fprintf(out, "# $d = number of members\n", 5 + 2 * ctl->nq);
04965         }
04966
04967         /* Set time interval... */
04968         t0 = t - 0.5 * ctl->dt_mod;
04969         t1 = t + 0.5 * ctl->dt_mod;
04970
04971         /* Init... */
04972         ens = GSL_NAN;
04973         n = 0;
04974
04975         /* Loop over air parcels... */
04976         for (ip = 0; ip < atm->np; ip++) {
04977
04978             /* Check time... */
04979             if (atm->time[ip] < t0 || atm->time[ip] > t1)
04980                 continue;
04981
04982             /* Check ensemble id... */
04983             if (atm->q[ctl->qnt_ens][ip] != ens) {
04984
04985                 /* Write results... */
04986                 if (n > 0) {
04987
04988                     /* Get mean position... */
04989                     xm[0] = xm[1] = xm[2] = 0;
04990                     for (i = 0; i < n; i++) {
04991                         xm[0] += x[i][0] / (double) n;
04992                         xm[1] += x[i][1] / (double) n;
04993                         xm[2] += x[i][2] / (double) n;
04994                     }
04995                     cart2geo(xm, &dummy, &lon, &lat);

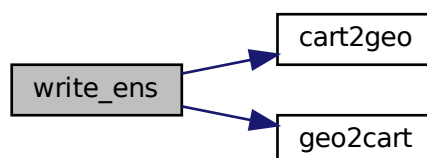
```

```

04996     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
04997         lat);
04998
04999     /* Get quantity statistics... */
05000     for (iq = 0; iq < ctl->nq; iq++) {
05001         fprintf(out, " ");
05002         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
05003     }
05004     for (iq = 0; iq < ctl->nq; iq++) {
05005         fprintf(out, " ");
05006         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
05007     }
05008     fprintf(out, " %zu\n", n);
05009 }
05010
05011 /* Init new ensemble... */
05012 ens = atm->q[ctl->qnt_ens][ip];
05013 n = 0;
05014 }
05015
05016 /* Save data... */
05017 p[n] = atm->p[ip];
05018 geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
05019 for (iq = 0; iq < ctl->nq; iq++)
05020     q[iq][n] = atm->q[iq][ip];
05021 if ((++n) >= NENS)
05022     ERRMSG("Too many data points!");
05023 }
05024
05025 /* Write results... */
05026 if (n > 0) {
05027     /* Get mean position... */
05028     xm[0] = xm[1] = xm[2] = 0;
05029     for (i = 0; i < n; i++) {
05030         xm[0] += x[i][0] / (double) n;
05031         xm[1] += x[i][1] / (double) n;
05032         xm[2] += x[i][2] / (double) n;
05033     }
05034     cart2geo(xm, &dummy, &lon, &lat);
05035     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
05036
05037     /* Get quantity statistics... */
05038     for (iq = 0; iq < ctl->nq; iq++) {
05039         fprintf(out, " ");
05040         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
05041     }
05042     for (iq = 0; iq < ctl->nq; iq++) {
05043         fprintf(out, " ");
05044         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
05045     }
05046     fprintf(out, " %zu\n", n);
05047 }
05048 }
05049
05050 /* Close file... */
05051 if (t == ctl->t_stop)
05052     fclose(out);
05053 }

```

Here is the call graph for this function:



5.21.3.50 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 5057 of file libtrac.c.

```

05063     {
05064
05065     FILE *in, *out;
05066
05067     char line[LEN];
05068
05069     static double mass[GX][GY][GZ], vmr[GX][GY][GZ], vmr_expl, vmr_impl,
05070     z[GZ], dz, lon[GX], dlon, lat[GY], dlat, area[GY], rho_air,
05071     press[GZ], temp, cd, t0, t1, r;
05072
05073     static int ip, ix, *ixs, iy, *iys, iz, *izs, np[GX][GY][GZ], year, mon, day,
05074     hour, min, sec;
05075
05076     /* Set timer... */
05077     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
05078
05079     /* Check dimensions... */
05080     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
05081         ERRMSG("Grid dimensions too large!");
05082
05083     /* Set grid box size... */
05084     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
05085     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
05086     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
05087
05088     /* Set vertical coordinates... */
05089     #pragma omp parallel for default(shared) private(iz)
05090     for (iz = 0; iz < ctl->grid_nz; iz++) {
05091         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
05092         press[iz] = P(z[iz]);
05093     }
05094
05095     /* Set horizontal coordinates... */
05096     for (ix = 0; ix < ctl->grid_nx; ix++)
05097         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
05098     #pragma omp parallel for default(shared) private(iy)
05099     for (iy = 0; iy < ctl->grid_ny; iy++) {
05100         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
05101         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05102             * cos(lat[iy] * M_PI / 180.);
05103     }
05104
05105     /* Set time interval for output... */
05106     t0 = t - 0.5 * ctl->dt_mod;
05107     t1 = t + 0.5 * ctl->dt_mod;
05108
05109     /* Initialize grid... */
05110     #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(3)
05111     for (ix = 0; ix < ctl->grid_nx; ix++)
05112         for (iy = 0; iy < ctl->grid_ny; iy++)
05113             for (iz = 0; iz < ctl->grid_nz; iz++) {
05114                 mass[ix][iy][iz] = 0;
05115                 vmr[ix][iy][iz] = 0;
05116                 np[ix][iy][iz] = 0;
05117             }
05118
05119     /* Allocate... */
05120     ALLOC(ixs, int,
05121         atm->np);
05122     ALLOC(iys, int,
05123         atm->np);
05124     ALLOC(izs, int,
05125         atm->np);
05126
05127     /* Get indices... */
05128     #pragma omp parallel for default(shared) private(ip)
05129     for (ip = 0; ip < atm->np; ip++) {
05130         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
05131         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
05132         izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05133         if (atm->time[ip] < t0 || atm->time[ip] > t1

```

```

05134         || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05135         || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05136         || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05137     izs[ip] = -1;
05138 }
05139
05140 /* Average data... */
05141 for (ip = 0; ip < atm->np; ip++)
05142     if (izs[ip] >= 0) {
05143         np[ixs[ip]][iys[ip]][izs[ip]]++;
05144         if (ctl->qnt_m >= 0)
05145             mass[ixs[ip]][iys[ip]][izs[ip]] += atm->q[ctl->qnt_m][ip];
05146         if (ctl->qnt_vmr >= 0)
05147             vmr[ixs[ip]][iys[ip]][izs[ip]] += atm->q[ctl->qnt_vmr][ip];
05148     }
05149
05150 /* Free... */
05151 free(ixs);
05152 free(iys);
05153 free(izs);
05154
05155 /* Check if gnuplot output is requested... */
05156 if (ctl->grid_gpfile[0] != '-') {
05157
05158     /* Write info... */
05159     LOG(1, "Plot grid data: %s.png", filename);
05160
05161     /* Create gnuplot pipe... */
05162     if (!(out = popen("gnuplot", "w")))
05163         ERRMSG("Cannot create pipe to gnuplot!");
05164
05165     /* Set plot filename... */
05166     fprintf(out, "set out \"%s.png\"\n", filename);
05167
05168     /* Set time string... */
05169     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05170     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05171             year, mon, day, hour, min);
05172
05173     /* Dump gnuplot file to pipe... */
05174     if (!(in = fopen(ctl->grid_gpfile, "r")))
05175         ERRMSG("Cannot open file!");
05176     while (fgets(line, LEN, in))
05177         fprintf(out, "%s", line);
05178     fclose(in);
05179 }
05180
05181 else {
05182
05183     /* Write info... */
05184     LOG(1, "Write grid data: %s", filename);
05185
05186     /* Create file... */
05187     if (!(out = fopen(filename, "w")))
05188         ERRMSG("Cannot create file!");
05189 }
05190
05191 /* Write header... */
05192 fprintf(out,
05193         "# $1 = time [s]\n"
05194         "# $2 = altitude [km]\n"
05195         "# $3 = longitude [deg]\n"
05196         "# $4 = latitude [deg]\n"
05197         "# $5 = surface area [km^2]\n"
05198         "# $6 = layer width [km]\n"
05199         "# $7 = number of particles [l]\n"
05200         "# $8 = column density (implicit) [kg/m^2]\n"
05201         "# $9 = volume mixing ratio (implicit) [ppv]\n"
05202         "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
05203
05204 /* Write data... */
05205 for (ix = 0; ix < ctl->grid_nx; ix++) {
05206     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
05207         fprintf(out, "\n");
05208     for (iy = 0; iy < ctl->grid_ny; iy++) {
05209         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
05210             fprintf(out, "\n");
05211         for (iz = 0; iz < ctl->grid_nz; iz++)
05212             if (!ctl->grid_sparse || mass[ix][iy][iz] > 0 || vmr[ix][iy][iz] > 0) {
05213
05214                 /* Calculate column density... */
05215                 if (ctl->qnt_m >= 0)
05216                     cd = mass[ix][iy][iz] / (1e6 * area[iy]);
05217                 else
05218                     cd = GSL_NAN;
05219
05220                 /* Calculate volume mixing ratio (implicit)... */

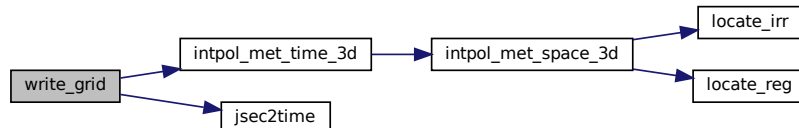
```

```

05221     if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05222         vmr_impl = 0;
05223         if (mass[ix][iy][iz] > 0) {
05224             /* Get temperature... */
05225             INTPOL_INIT;
05226             intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05227                             lon[ix], lat[iy], &temp, ci, cw, 1);
05228
05229             /* Calculate density of air... */
05230             rho_air = 100. * press[iz] / (RA * temp);
05231
05232             /* Calculate volume mixing ratio... */
05233             vmr_impl = MA / ctl->molmass * mass[ix][iy][iz]
05234                 / (rho_air * 1e6 * area[iy] * 1e3 * dz);
05235         }
05236     } else
05237     {
05238         vmr_impl = GSL_NAN;
05239
05240         /* Calculate volume mixing ratio (explicit)... */
05241         if (ctl->qnt_vmr >= 0 && np[ix][iy][iz] > 0)
05242             vmr_expl = vmr[ix][iy][iz] / np[ix][iy][iz];
05243         else
05244             vmr_expl = GSL_NAN;
05245
05246         /* Write output... */
05247         fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", t, z[iz],
05248             lon[ix], lat[iy], area[iy], dz, np[ix][iy][iz], cd,
05249             vmr_impl, vmr_expl);
05250     }
05251 }
05252 }
05253
05254 /* Close file... */
05255 fclose(out);
05256 }

```

Here is the call graph for this function:



5.21.3.51 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 5260 of file libtrac.c.

```

05266     {
05267
05268         static FILE *in, *out;
05269
05270         static char line[LEN];
05271
05272         static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rt_old,
05273             rz, rlon, rlat, robs, t0, t1, area[GY], dz, dlon, dlat,
05274             lon[GX], lat[GY], z[GZ], press[GZ], temp, rho_air, vmr, h2o, o3;

```

```

05275
05276 static int obscount[GX][GY], ip, ix, iy, iz, okay;
05277
05278 /* Set timer... */
05279 SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
05280
05281 /* Init... */
05282 if (t == ctl->t_start) {
05283
05284     /* Check quantity index for mass... */
05285     if (ctl->qnt_m < 0)
05286         ERRMSG("Need quantity mass!");
05287
05288     /* Check dimensions... */
05289     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
05290         ERRMSG("Grid dimensions too large!");
05291
05292     /* Check molar mass... */
05293     if (ctl->molmass <= 0)
05294         ERRMSG("Specify molar mass!");
05295
05296     /* Open observation data file... */
05297     LOG(1, "Read profile observation data: %s", ctl->prof_obsfile);
05298     if (!in = fopen(ctl->prof_obsfile, "r"))
05299         ERRMSG("Cannot open file!");
05300
05301     /* Initialize time for file input... */
05302     rt_old = -1e99;
05303
05304     /* Create new output file... */
05305     LOG(1, "Write profile data: %s", filename);
05306     if (!out = fopen(filename, "w"))
05307         ERRMSG("Cannot create file!");
05308
05309     /* Write header... */
05310     fprintf(out,
05311         "# $1 = time [s]\n"
05312         "# $2 = altitude [km]\n"
05313         "# $3 = longitude [deg]\n"
05314         "# $4 = latitude [deg]\n"
05315         "# $5 = pressure [hPa]\n"
05316         "# $6 = temperature [K]\n"
05317         "# $7 = volume mixing ratio [ppv]\n"
05318         "# $8 = H2O volume mixing ratio [ppv]\n"
05319         "# $9 = O3 volume mixing ratio [ppv]\n"
05320         "# $10 = observed BT index [K]\n"
05321         "# $11 = number of observations\n");
05322
05323     /* Set grid box size... */
05324     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
05325     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
05326     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
05327
05328     /* Set vertical coordinates... */
05329     for (iz = 0; iz < ctl->prof_nz; iz++) {
05330         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
05331         press[iz] = P(z[iz]);
05332     }
05333
05334     /* Set horizontal coordinates... */
05335     for (ix = 0; ix < ctl->prof_nx; ix++)
05336         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
05337     for (iy = 0; iy < ctl->prof_ny; iy++) {
05338         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
05339         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05340             * cos(lat[iy] * M_PI / 180.);
05341     }
05342 }
05343
05344 /* Set time interval... */
05345 t0 = t - 0.5 * ctl->dt_mod;
05346 t1 = t + 0.5 * ctl->dt_mod;
05347
05348 /* Initialize... */
05349 #pragma omp parallel for default(shared) private(ix,iy,iz) collapse(2)
05350 for (ix = 0; ix < ctl->prof_nx; ix++)
05351     for (iy = 0; iy < ctl->prof_ny; iy++) {
05352         obsmean[ix][iy] = 0;
05353         obscount[ix][iy] = 0;
05354         for (iz = 0; iz < ctl->prof_nz; iz++)
05355             mass[ix][iy][iz] = 0;
05356     }
05357
05358 /* Read observation data... */
05359 while (fgets(line, LEN, in)) {
05360
05361     /* Read data... */

```

```

05362     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
05363         5)
05364         continue;
05365
05366     /* Check time... */
05367     if (rt < t0)
05368         continue;
05369     if (rt > t1)
05370         break;
05371     if (rt < rt_old)
05372         ERRMSG("Time must be ascending!");
05373     rt_old = rt;
05374
05375     /* Check observation data... */
05376     if (!isfinite(robs))
05377         continue;
05378
05379     /* Calculate indices... */
05380     ix = (int) ((rln - ctl->prof_lon0) / dlon);
05381     iy = (int) ((rln - ctl->prof_lat0) / dlat);
05382
05383     /* Check indices... */
05384     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
05385         continue;
05386
05387     /* Get mean observation index... */
05388     obsmean[ix][iy] += robs;
05389
05390     /* Count observations... */
05391     obscount[ix][iy]++;
05392 }
05393
05394 /* Analyze model data... */
05395 for (ip = 0; ip < atm->np; ip++) {
05396
05397     /* Check time... */
05398     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05399         continue;
05400
05401     /* Get indices... */
05402     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
05403     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
05404     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
05405
05406     /* Check indices... */
05407     if (ix < 0 || ix >= ctl->prof_nx ||
05408         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
05409         continue;
05410
05411     /* Get total mass in grid cell... */
05412     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
05413 }
05414
05415 /* Extract profiles... */
05416 for (ix = 0; ix < ctl->prof_nx; ix++)
05417     for (iy = 0; iy < ctl->prof_ny; iy++)
05418         if (obscount[ix][iy] >= 1) {
05419
05420             /* Check profile... */
05421             okay = 0;
05422             for (iz = 0; iz < ctl->prof_nz; iz++)
05423                 if (mass[ix][iy][iz] > 0) {
05424                     okay = 1;
05425                     break;
05426                 }
05427             if (!okay)
05428                 continue;
05429
05430             /* Write output... */
05431             fprintf(out, "\n");
05432
05433             /* Loop over altitudes... */
05434             for (iz = 0; iz < ctl->prof_nz; iz++) {
05435
05436                 /* Get pressure and temperature... */
05437                 INTPOL_INIT;
05438                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05439                     lon[ix], lat[iy], &temp, ci, cw, 1);
05440                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
05441                     lon[ix], lat[iy], &h2o, ci, cw, 0);
05442                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
05443                     lon[ix], lat[iy], &o3, ci, cw, 0);
05444
05445                 /* Calculate volume mixing ratio... */
05446                 rho_air = 100. * press[iz] / (RA * temp);
05447                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
05448                     / (rho_air * area[iy] * dz * 1e9);

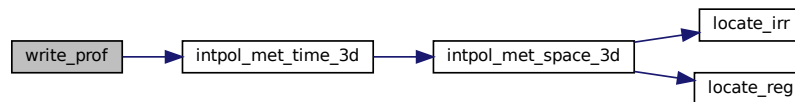
```

```

05449
05450     /* Write output... */
05451     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
05452            t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
05453            obsmean[ix][iy] / obscount[ix][iy], obscount[ix][iy]);
05454     }
05455 }
05456
05457 /* Close files... */
05458 if (t == ctl->t_stop) {
05459     fclose(in);
05460     fclose(out);
05461 }
05462 }

```

Here is the call graph for this function:



5.21.3.52 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 5466 of file libtrac.c.

```

05472     {
05473
05474     static FILE *in, *out;
05475
05476     static char line[LEN];
05477
05478     static double area, dlat, rmax2, t0, t1, rt, rt_old, rz, rlon, rlat, robs;
05479
05480     /* Set timer... */
05481     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
05482
05483     /* Init... */
05484     if (t == ctl->t_start) {
05485
05486         /* Open observation data file... */
05487         LOG(1, "Read sample observation data: %s", ctl->sample_obsfile);
05488         if (!(in = fopen(ctl->sample_obsfile, "r")))
05489             ERRMSG("Cannot open file!");
05490
05491         /* Initialize time for file input... */
05492         rt_old = -1e99;
05493
05494         /* Create new file... */
05495         LOG(1, "Write sample data: %s", filename);
05496         if (!(out = fopen(filename, "w")))
05497             ERRMSG("Cannot create file!");
05498
05499         /* Write header... */
05500         fprintf(out,
05501            "# $1 = time [s]\n"
05502            "# $2 = altitude [km]\n"
05503            "# $3 = longitude [deg]\n"

```



```

05504         "# $4 = latitude [deg]\n"
05505         "# $5 = surface area [km^2]\n"
05506         "# $6 = layer width [km]\n"
05507         "# $7 = number of particles [1]\n"
05508         "# $8 = column density [kg/m^2]\n"
05509         "# $9 = volume mixing ratio [ppv]\n"
05510         "# $10 = observed BT index [K]\n\n");
05511
05512     /* Set latitude range, squared radius, and area... */
05513     dlat = DY2DEG(ctl->sample_dx);
05514     rmax2 = SQR(ctl->sample_dx);
05515     area = M_PI * rmax2;
05516 }
05517
05518 /* Set time interval for output... */
05519 t0 = t - 0.5 * ctl->dt_mod;
05520 t1 = t + 0.5 * ctl->dt_mod;
05521
05522 /* Read observation data... */
05523 while (fgets(line, LEN, in)) {
05524
05525     /* Read data... */
05526     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rlat, &robs) !=
05527         5)
05528         continue;
05529
05530     /* Check time... */
05531     if (rt < t0)
05532         continue;
05533     if (rt < rt_old)
05534         ERRMSG("Time must be ascending!");
05535     rt_old = rt;
05536
05537     /* Calculate Cartesian coordinates... */
05538     double x0[3];
05539     geo2cart(0, rln, rlat, x0);
05540
05541     /* Set pressure range... */
05542     double rp = P(rz);
05543     double ptop = P(rz + ctl->sample_dz);
05544     double pbot = P(rz - ctl->sample_dz);
05545
05546     /* Init... */
05547     double mass = 0;
05548     int np = 0;
05549
05550     /* Loop over air parcels... */
05551     #pragma omp parallel for default(shared) reduction(+:mass,np)
05552     for (int ip = 0; ip < atm->np; ip++) {
05553
05554         /* Check time... */
05555         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05556             continue;
05557
05558         /* Check latitude... */
05559         if (fabs(rlat - atm->lat[ip]) > dlat)
05560             continue;
05561
05562         /* Check horizontal distance... */
05563         double x1[3];
05564         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05565         if (DIST2(x0, x1) > rmax2)
05566             continue;
05567
05568         /* Check pressure... */
05569         if (ctl->sample_dz > 0)
05570             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
05571                 continue;
05572
05573         /* Add mass... */
05574         if (ctl->qnt_m >= 0)
05575             mass += atm->q[ctl->qnt_m][ip];
05576         np++;
05577     }
05578
05579     /* Calculate column density... */
05580     double cd = mass / (1e6 * area);
05581
05582     /* Calculate volume mixing ratio... */
05583     double vmr = 0;
05584     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
05585         if (mass > 0) {
05586
05587             /* Get temperature... */
05588             double temp;
05589             INTPOL_INIT;
05590             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt, rp,

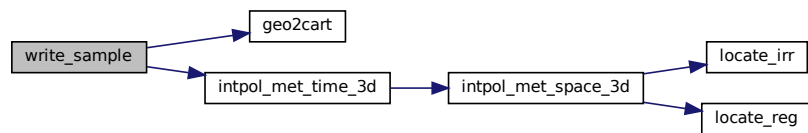
```

```

05591             rlon, rlat, &temp, ci, cw, 1);
05592
05593     /* Calculate density of air... */
05594     double rho_air = 100. * rp / (RA * temp);
05595
05596     /* Calculate volume mixing ratio... */
05597     vmr = MA / ctl->molmass * mass
05598           / (rho_air * 1e6 * area * 1e3 * ctl->sample_dz);
05599 }
05600 } else
05601     vmr = GSL_NAN;
05602
05603 /* Write output... */
05604 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt, rz, rlon, rlat,
05605         area, ctl->sample_dz, np, cd, vmr, robs);
05606
05607 /* Check time... */
05608 if (rt >= tl)
05609     break;
05610 }
05611
05612 /* Close files... */
05613 if (t == ctl->t_stop) {
05614     fclose(in);
05615     fclose(out);
05616 }
05617 }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 5621 of file libtrac.c.

```

05625     {
05626
05627     static FILE *out;
05628
05629     static double rmax2, t0, t1, x0[3], x1[3];
05630
05631     /* Set timer... */
05632     SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
05633
05634     /* Init... */
05635     if (t == ctl->t_start) {
05636
05637         /* Write info... */
05638         LOG(1, "Write station data: %s", filename);
05639
05640         /* Create new file... */
05641         if (!(out = fopen(filename, "w")))
05642             ERRMSG("Cannot create file!");
05643
05644         /* Write header... */
05645         fprintf(out,

```

```

05646         "# $1 = time [s]\n"
05647         "# $2 = altitude [km]\n"
05648         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05649     for (int iq = 0; iq < ctl->nq; iq++)
05650         fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
05651             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05652     fprintf(out, "\n");
05653
05654     /* Set geolocation and search radius... */
05655     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
05656     rmax2 = SQR(ctl->stat_r);
05657 }
05658
05659 /* Set time interval for output... */
05660 t0 = t - 0.5 * ctl->dt_mod;
05661 t1 = t + 0.5 * ctl->dt_mod;
05662
05663 /* Loop over air parcels... */
05664 for (int ip = 0; ip < atm->np; ip++) {
05665
05666     /* Check time... */
05667     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05668         continue;
05669
05670     /* Check time range for station output... */
05671     if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
05672         continue;
05673
05674     /* Check station flag... */
05675     if (ctl->qnt_stat >= 0)
05676         if (atm->q[ctl->qnt_stat][ip])
05677             continue;
05678
05679     /* Get Cartesian coordinates... */
05680     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
05681
05682     /* Check horizontal distance... */
05683     if (DIST2(x0, x1) > rmax2)
05684         continue;
05685
05686     /* Set station flag... */
05687     if (ctl->qnt_stat >= 0)
05688         atm->q[ctl->qnt_stat][ip] = 1;
05689
05690     /* Write data... */
05691     fprintf(out, "%.2f %g %g %g",
05692         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
05693     for (int iq = 0; iq < ctl->nq; iq++) {
05694         fprintf(out, " ");
05695         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05696     }
05697     fprintf(out, "\n");
05698 }
05699
05700 /* Close file... */
05701 if (t == ctl->t_stop)
05702     fclose(out);
05703 }

```

Here is the call graph for this function:



5.22 libtrac.h

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by

```

```

00006 the Free Software Foundation, either version 3 of the License, or
00007 (at your option) any later version.
00008
00009 MPTRAC is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU General Public License for more details.
00013
00014 You should have received a copy of the GNU General Public License
00015 along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2013–2021 Forschungszentrum Juelich GmbH
00018 */
00019
00036 #ifndef LIBTRAC_H
00037 #define LIBTRAC_H
00038
00039 /* -----
00040 Includes...
00041 ----- */
00042
00043 #include <ctype.h>
00044 #include <gsl/gsl_fft_complex.h>
00045 #include <gsl/gsl_math.h>
00046 #include <gsl/gsl_randist.h>
00047 #include <gsl/gsl_rng.h>
00048 #include <gsl/gsl_sort.h>
00049 #include <gsl/gsl_spline.h>
00050 #include <gsl/gsl_statistics.h>
00051 #include <math.h>
00052 #include <netcdf.h>
00053 #include <omp.h>
00054 #include <stdio.h>
00055 #include <stdlib.h>
00056 #include <string.h>
00057 #include <time.h>
00058 #include <sys/time.h>
00059
00060 #ifdef MPI
00061 #include "mpi.h"
00062 #endif
00063
00064 #ifdef _OPENACC
00065 #include "openacc.h"
00066 #include "curand.h"
00067 #endif
00068
00069 /* -----
00070 Constants...
00071 ----- */
00072
00074 #define CPD 1003.5
00075
00077 #define EPS (MH2O / MA)
00078
00080 #define G0 9.80665
00081
00083 #define H0 7.0
00084
00086 #define LV 2501000.
00087
00089 #define KB 1.3806504e-23
00090
00092 #define MA 28.9644
00093
00095 #define MH2O 18.01528
00096
00098 #define MO3 48.00
00099
00101 #define P0 1013.25
00102
00104 #define RA (1e3 * RI / MA)
00105
00107 #define RE 6367.421
00108
00110 #define RI 8.3144598
00111
00113 #define T0 273.15
00114
00115 /* -----
00116 Dimensions...
00117 ----- */
00118
00120 #ifndef LEN
00121 #define LEN 5000
00122 #endif
00123

```

```

00125 #ifndef NP
00126 #define NP 10000000
00127 #endif
00128
00130 #ifndef NQ
00131 #define NQ 15
00132 #endif
00133
00135 #ifndef EP
00136 #define EP 140
00137 #endif
00138
00140 #ifndef EX
00141 #define EX 1201
00142 #endif
00143
00145 #ifndef EY
00146 #define EY 601
00147 #endif
00148
00150 #ifndef GX
00151 #define GX 720
00152 #endif
00153
00155 #ifndef GY
00156 #define GY 360
00157 #endif
00158
00160 #ifndef GZ
00161 #define GZ 100
00162 #endif
00163
00165 #ifndef NENS
00166 #define NENS 2000
00167 #endif
00168
00170 #ifndef NTHREADS
00171 #define NTHREADS 512
00172 #endif
00173
00174 /* -----
00175     Macros...
00176 ----- */
00177
00179 #ifdef _OPENACC
00180 #define ALLOC(ptr, type, n) \
00181     if(acc_get_num_devices(acc_device_nvidia) <= 0) \
00182         ERRMSG("Not running on a GPU device!"); \
00183     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00184         ERRMSG("Out of memory!");
00185 #else
00186 #define ALLOC(ptr, type, n) \
00187     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00188         ERRMSG("Out of memory!");
00189 #endif
00190
00192 #define DEG2DX(dlon, lat) \
00193     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00194
00196 #define DEG2DY(dlat) \
00197     ((dlat) * M_PI * RE / 180.)
00198
00200 #define DP2DZ(dp, p) \
00201     (- (dp) * H0 / (p))
00202
00204 #define DX2DEG(dx, lat) \
00205     (((lat) < -89.999 || (lat) > 89.999) ? 0 \
00206      : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI)))
00207
00209 #define DY2DEG(dy) \
00210     ((dy) * 180. / (M_PI * RE))
00211
00213 #define DZ2DP(dz, p) \
00214     (- (dz) * (p) / H0)
00215
00217 #define DIST(a, b) \
00218     sqrt(DIST2(a, b))
00219
00221 #define DIST2(a, b) \
00222     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00223
00225 #define DOTP(a, b) \
00226     (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00227
00229 #define FMOD(x, y) \
00230     ((x) - (int) ((x) / (y)) * (y))
00231

```

```

00233 #define FREAD(ptr, type, size, out) {
00234     if(fread(ptr, sizeof(type), size, out)!=size)
00235         ERRMSG("Error while reading!");
00236 }
00237
00239 #define FWRITE(ptr, type, size, out) {
00240     if(fwrite(ptr, sizeof(type), size, out)!=size)
00241         ERRMSG("Error while writing!");
00242 }
00243
00245 #define INTPOL_INIT
00246     double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};
00247
00249 #define INTPOL_2D(var, init)
00250     intpol_met_time_2d(met0->var, met1->var,
00251         atm->time[ip], atm->lon[ip], atm->lat[ip],
00252         &var, ci, cw, init);
00253
00255 #define INTPOL_3D(var, init)
00256     intpol_met_time_3d(met0, met0->var, met1, met1->var,
00257         atm->time[ip], atm->p[ip],
00258         atm->lon[ip], atm->lat[ip],
00259         &var, ci, cw, init);
00260
00262 #define INTPOL_SPACE_ALL(p, lon, lat) {
00263     intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1);
00264     intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0);
00265     intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0);
00266     intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0);
00267     intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0);
00268     intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0);
00269     intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0);
00270     intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0);
00271     intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0);
00272     intpol_met_space_3d(met, met->iwc, p, lon, lat, &iwc, ci, cw, 0);
00273     intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
00274     intpol_met_space_2d(met, met->ts, lon, lat, &ts, ci, cw, 0);
00275     intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0);
00276     intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0);
00277     intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0);
00278     intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0);
00279     intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
00280     intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0);
00281     intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0);
00282     intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0);
00283     intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0);
00284     intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0);
00285     intpol_met_space_2d(met, met->c1, lon, lat, &c1, ci, cw, 0);
00286     intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0);
00287     intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0);
00288     intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0);
00289     intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0);
00290     intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0);
00291 }
00292
00294 #define INTPOL_TIME_ALL(time, p, lon, lat) {
00295     intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1);
00296     intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0);
00297     intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0);
00298     intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0);
00299     intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0);
00300     intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0);
00301     intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0);
00302     intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0);
00303     intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0);
00304     intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0);
00305     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0);
00306     intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0);
00307     intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0);
00308     intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0);
00309     intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0);
00310     intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0);
00311     intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0);
00312     intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0);
00313     intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0);
00314     intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0);
00315     intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0);
00316     intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0);
00317     intpol_met_time_2d(met0, met0->c1, met1, met1->c1, time, lon, lat, &c1, ci, cw, 0);
00318     intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0);
00319     intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0);
00320     intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0);
00321     intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0);
00322     intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0);
00323 }
00324
00326 #define LAPSE(pl, t1, p2, t2)

```

```

00327     (1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1))          \
00328     * ((p2) + (p1)) / ((p2) - (p1)))
00329
00331 #define LIN(x0, y0, x1, y1, x)                                \
00332     ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00333
00335 #define NC(cmd) {                                              \
00336     int nc_result=(cmd);                                       \
00337     if(nc_result!=NC_NOERR)                                    \
00338         ERRMSG("%s", nc_strerror(nc_result));                 \
00339 }
00340
00342 #define NN(x0, y0, x1, y1, x)                                  \
00343     (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))
00344
00346 #define NORM(a) \
00347     sqrt(DOTP(a, a))
00348
00350 #define P(z) \
00351     (P0 * exp(-(z) / H0))
00352
00354 #define PSAT(t) \
00355     (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))
00356
00358 #define PSICE(t) \
00359     (0.01 * pow(10., -2663.5 / (t) + 12.537))
00360
00362 #define PW(p, h2o) \
00363     ((p) * GSL_MAX((h2o), 0.1e-6)) \
00364     / (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6))
00365
00367 #define RH(p, t, h2o) \
00368     (PW(p, h2o) / PSAT(t) * 100.)
00369
00371 #define RHICE(p, t, h2o) \
00372     (PW(p, h2o) / PSICE(t) * 100.)
00373
00375 #define SET_ATM(qnt, val) \
00376     if (ctl->qnt >= 0) \
00377         atm->q[ctl->qnt][ip] = val;
00378
00380 #define SET_QNT(qnt, name, unit) \
00381     if (strcasecmp(ctl->qnt_name[iq], name) == 0) { \
00382         ctl->qnt = iq; \
00383         sprintf(ctl->qnt_unit[iq], unit); \
00384     } else
00385
00387 #define SH(h2o) \
00388     (EPS * GSL_MAX((h2o), 0.1e-6))
00389
00391 #define SQR(x) \
00392     ((x)*(x))
00393
00395 #define TDEW(p, h2o) \
00396     (T0 + 243.12 * log(PW((p), (h2o)) / 6.112) \
00397     / (17.62 - log(PW((p), (h2o)) / 6.112)))
00398
00400 #define TICE(p, h2o) \
00401     (-2663.5 / (log10(100. * PW((p), (h2o))) - 12.537))
00402
00404 #define THETA(p, t) \
00405     ((t) * pow(1000. / (p), 0.286))
00406
00408 #define THETA_VIRT(p, t, h2o) \
00409     (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))
00410
00412 #define TOK(line, tok, format, var) { \
00413     if((tok)=strtok((line), " \t")) { \
00414         if(sscanf(tok, format, &(var))!=1) continue; \
00415     } else ERRMSG("Error while reading!"); \
00416 }
00417
00419 #define TVIRT(t, h2o) \
00420     ((t) * (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
00421
00423 #define Z(p) \
00424     (H0 * log(P0 / (p)))
00425
00427 #define ZDIFF(lnp0, t0, h2o0, lnp1, t1, h2o1) \
00428     (RI / MA / G0 * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1))) \
00429     * ((lnp0) - (lnp1)))
00430
00432 #define ZETA(ps, p, t) \
00433     ((p) / (ps) <= 0.3 ? 1. : \
00434     sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3))) \
00435     * THETA((p), (t))
00436

```

```

00437 /* -----
00438     Log messages...
00439     ----- */
00440
00442 #ifndef LOGLEV
00443 #define LOGLEV 2
00444 #endif
00445
00447 #define LOG(level, ...) {
00448     if(level >= 2)
00449         printf(" ");
00450     if(level <= LOGLEV) {
00451         printf(__VA_ARGS__);
00452         printf("\n");
00453     }
00454 }
00455
00457 #define WARN(...) {
00458     printf("\nWarning (%s, %s, l%d): ", __FILE__, __func__, __LINE__);
00459     LOG(0, __VA_ARGS__);
00460 }
00461
00463 #define ERRMSG(...) {
00464     printf("\nError (%s, %s, l%d): ", __FILE__, __func__, __LINE__);
00465     LOG(0, __VA_ARGS__);
00466     exit(EXIT_FAILURE);
00467 }
00468
00470 #define PRINT(format, var)
00471     printf("Print (%s, %s, l%d): %s= \"format\"\n",
00472         __FILE__, __func__, __LINE__, #var, var);
00473
00474 /* -----
00475     Timers...
00476     ----- */
00477
00479 #define NTIMER 100
00480
00482 #define PRINT_TIMERS \
00483     timer("END", "END", 1);
00484
00486 #define SELECT_TIMER(id, group, color) {
00487     NVTX_POP;
00488     NVTX_PUSH(id, color);
00489     timer(id, group, 0);
00490 }
00491
00493 #define START_TIMERS \
00494     NVTX_PUSH("START", NVTX_CPU);
00495
00497 #define STOP_TIMERS \
00498     NVTX_POP;
00499
00500 /* -----
00501     NVIDIA Tools Extension (NVTX)...
00502     ----- */
00503
00504 #ifdef NVTX
00505 #include "nvToolsExt.h"
00506
00508 #define NVTX_CPU 0xFFADD8E6
00509
00511 #define NVTX_GPU 0xFF00008B
00512
00514 #define NVTX_H2D 0xFFFFFFFF00
00515
00517 #define NVTX_D2H 0xFFFF8800
00518
00520 #define NVTX_READ 0xFFFFCCCB
00521
00523 #define NVTX_WRITE 0xFF8B0000
00524
00526 #define NVTX_PUSH(range_title, range_color) {
00527     nvtxEventAttributes_t eventAttrib = {0};
00528     eventAttrib.version = NVTX_VERSION;
00529     eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;
00530     eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII;
00531     eventAttrib.colorType = NVTX_COLOR_ARGB;
00532     eventAttrib.color = range_color;
00533     eventAttrib.message.ascii = range_title;
00534     nvtxRangePushEx(&eventAttrib);
00535 }
00536
00538 #define NVTX_POP {
00539     nvtxRangePop();
00540 }
00541 #else

```



```

00542
00543 /* Empty definitions of NVTX_PUSH and NVTX_POP... */
00544 #define NVTX_PUSH(range_title, range_color) {}
00545 #define NVTX_POP {}
00546 #endif
00547
00548 /* -----
00549   Structs...
00550   ----- */
00551
00552 typedef struct {
00553     size_t  chunkszhint;
00554
00555     int  read_mode;
00556
00557     int  nq;
00558
00559     char qnt_name[NQ][LEN];
00560
00561     char qnt_unit[NQ][LEN];
00562
00563     char qnt_format[NQ][LEN];
00564
00565     int  qnt_ens;
00566
00567     int  qnt_stat;
00568
00569     int  qnt_m;
00570
00571     int  qnt_vmr;
00572
00573     int  qnt_rho;
00574
00575     int  qnt_r;
00576
00577     int  qnt_ps;
00578
00579     int  qnt_ts;
00580
00581     int  qnt_zs;
00582
00583     int  qnt_us;
00584
00585     int  qnt_vs;
00586
00587     int  qnt_pbl;
00588
00589     int  qnt_pt;
00590
00591     int  qnt_tt;
00592
00593     int  qnt_zt;
00594
00595     int  qnt_h2ot;
00596
00597     int  qnt_z;
00598
00599     int  qnt_p;
00600
00601     int  qnt_t;
00602
00603     int  qnt_u;
00604
00605     int  qnt_v;
00606
00607     int  qnt_w;
00608
00609     int  qnt_h2o;
00610
00611     int  qnt_o3;
00612
00613     int  qnt_lwc;
00614
00615     int  qnt_iwc;
00616
00617     int  qnt_pct;
00618
00619     int  qnt_pcb;
00620
00621     int  qnt_cl;
00622
00623     int  qnt_plcl;
00624
00625     int  qnt_plfc;
00626
00627     int  qnt_pel;

```

```
00668
00670     int qnt_cape;
00671
00673     int qnt_cin;
00674
00676     int qnt_hno3;
00677
00679     int qnt_oh;
00680
00682     int qnt_psat;
00683
00685     int qnt_psice;
00686
00688     int qnt_pw;
00689
00691     int qnt_sh;
00692
00694     int qnt_rh;
00695
00697     int qnt_rhice;
00698
00700     int qnt_theta;
00701
00703     int qnt_zeta;
00704
00706     int qnt_tvirt;
00707
00709     int qnt_lapse;
00710
00712     int qnt_vh;
00713
00715     int qnt_vz;
00716
00718     int qnt_pv;
00719
00721     int qnt_tdew;
00722
00724     int qnt_tice;
00725
00727     int qnt_tsts;
00728
00730     int qnt_tnat;
00731
00733     int direction;
00734
00736     double t_start;
00737
00739     double t_stop;
00740
00742     double dt_mod;
00743
00745     char metbase[LEN];
00746
00748     double dt_met;
00749
00751     int met_dx;
00752
00754     int met_dy;
00755
00757     int met_dp;
00758
00760     int met_sx;
00761
00763     int met_sy;
00764
00766     int met_sp;
00767
00769     double met_detrend;
00770
00772     int met_np;
00773
00775     double met_p[EP];
00776
00778     int met_geopot_sx;
00779
00781     int met_geopot_sy;
00782
00785     int met_tropo;
00786
00788     double met_tropo_lapse;
00789
00791     int met_tropo_nlev;
00792
00794     double met_tropo_lapse_sep;
00795
00797     int met_tropo_nlev_sep;
00798
```

```
00800 double met_tropo_pv;
00801
00803 double met_tropo_theta;
00804
00806 int met_tropo_spline;
00807
00809 double met_cloud;
00810
00812 double met_dt_out;
00813
00815 int met_cache;
00816
00819 int isosurf;
00820
00822 char balloon[LEN];
00823
00825 int advect;
00826
00828 int reflect;
00829
00831 double turb_dx_trop;
00832
00834 double turb_dx_strat;
00835
00837 double turb_dz_trop;
00838
00840 double turb_dz_strat;
00841
00843 double turb_mesox;
00844
00846 double turb_mesoz;
00847
00849 double conv_cape;
00850
00852 double conv_cin;
00853
00855 double conv_wmax;
00856
00858 double conv_wcape;
00859
00861 double conv_dt;
00862
00864 int conv_mix_bot;
00865
00867 int conv_mix_top;
00868
00870 double bound_mass;
00871
00873 double bound_vmr;
00874
00876 double bound_lat0;
00877
00879 double bound_lat1;
00880
00882 double bound_p0;
00883
00885 double bound_p1;
00886
00888 double bound_dps;
00889
00891 char species[LEN];
00892
00894 double molmass;
00895
00897 double tdec_trop;
00898
00900 double tdec_strat;
00901
00903 int oh_chem_reaction;
00904
00906 double oh_chem[4];
00907
00909 double dry_depo[1];
00910
00912 double wet_depo[8];
00913
00915 double psc_h2o;
00916
00918 double psc_hno3;
00919
00921 char atm_basename[LEN];
00922
00924 char atm_gpfile[LEN];
00925
00927 double atm_dt_out;
00928
00930 int atm_filter;
```

```
00931
00933     int atm_stride;
00934
00936     int atm_type;
00937
00939     char csi_basename[LEN];
00940
00942     double csi_dt_out;
00943
00945     char csi_obsfile[LEN];
00946
00948     double csi_obsmin;
00949
00951     double csi_modmin;
00952
00954     int csi_nz;
00955
00957     double csi_z0;
00958
00960     double csi_z1;
00961
00963     int csi_nx;
00964
00966     double csi_lon0;
00967
00969     double csi_lon1;
00970
00972     int csi_ny;
00973
00975     double csi_lat0;
00976
00978     double csi_lat1;
00979
00981     char grid_basename[LEN];
00982
00984     char grid_gpfile[LEN];
00985
00987     double grid_dt_out;
00988
00990     int grid_sparse;
00991
00993     int grid_nz;
00994
00996     double grid_z0;
00997
00999     double grid_z1;
01000
01002     int grid_nx;
01003
01005     double grid_lon0;
01006
01008     double grid_lon1;
01009
01011     int grid_ny;
01012
01014     double grid_lat0;
01015
01017     double grid_lat1;
01018
01020     char prof_basename[LEN];
01021
01023     char prof_obsfile[LEN];
01024
01026     int prof_nz;
01027
01029     double prof_z0;
01030
01032     double prof_z1;
01033
01035     int prof_nx;
01036
01038     double prof_lon0;
01039
01041     double prof_lon1;
01042
01044     int prof_ny;
01045
01047     double prof_lat0;
01048
01050     double prof_lat1;
01051
01053     char ens_basename[LEN];
01054
01056     char sample_basename[LEN];
01057
01059     char sample_obsfile[LEN];
01060
```

```
01062 double sample_dx;
01063
01065 double sample_dz;
01066
01068 char stat_basename[LEN];
01069
01071 double stat_lon;
01072
01074 double stat_lat;
01075
01077 double stat_r;
01078
01080 double stat_t0;
01081
01083 double stat_t1;
01084
01085 } ctl_t;
01086
01088 typedef struct {
01089
01091 int np;
01092
01094 double time[NP];
01095
01097 double p[NP];
01098
01100 double lon[NP];
01101
01103 double lat[NP];
01104
01106 double q[NQ][NP];
01107
01108 } atm_t;
01109
01111 typedef struct {
01112
01114 double tsig[EX][EY][EP];
01115
01117 float uvwsig[EX][EY][EP][3];
01118
01120 float uvwp[NP][3];
01121
01123 double iso_var[NP];
01124
01126 double iso_ps[NP];
01127
01129 double iso_ts[NP];
01130
01132 int iso_n;
01133
01134 } cache_t;
01135
01137 typedef struct {
01138
01140 double time;
01141
01143 int nx;
01144
01146 int ny;
01147
01149 int np;
01150
01152 double lon[EX];
01153
01155 double lat[EY];
01156
01158 double p[EP];
01159
01161 float ps[EX][EY];
01162
01164 float ts[EX][EY];
01165
01167 float zs[EX][EY];
01168
01170 float us[EX][EY];
01171
01173 float vs[EX][EY];
01174
01176 float pbl[EX][EY];
01177
01179 float pt[EX][EY];
01180
01182 float tt[EX][EY];
01183
01185 float zt[EX][EY];
01186
01188 float h2ot[EX][EY];
```

```

01189
01191 float pct[EX][EY];
01192
01194 float pcb[EX][EY];
01195
01197 float cl[EX][EY];
01198
01200 float plcl[EX][EY];
01201
01203 float plfc[EX][EY];
01204
01206 float pel[EX][EY];
01207
01209 float cape[EX][EY];
01210
01212 float cin[EX][EY];
01213
01215 float z[EX][EY][EP];
01216
01218 float t[EX][EY][EP];
01219
01221 float u[EX][EY][EP];
01222
01224 float v[EX][EY][EP];
01225
01227 float w[EX][EY][EP];
01228
01230 float pv[EX][EY][EP];
01231
01233 float h2o[EX][EY][EP];
01234
01236 float o3[EX][EY][EP];
01237
01239 float lwc[EX][EY][EP];
01240
01242 float iwc[EX][EY][EP];
01243
01245 float pl[EX][EY][EP];
01246
01247 } met_t;
01248
01249 /* -----
01250 Functions...
01251 ----- */
01252
01254 void cart2geo(
01255     double *x,
01256     double *z,
01257     double *lon,
01258     double *lat);
01259
01261 #ifdef _OPENACC
01262 #pragma acc routine (check_finite)
01263 #endif
01264 int check_finite(
01265     const double x);
01266
01268 #ifdef _OPENACC
01269 #pragma acc routine (clim_hno3)
01270 #endif
01271 double clim_hno3(
01272     double t,
01273     double lat,
01274     double p);
01275
01277 #ifdef _OPENACC
01278 #pragma acc routine (clim_oh)
01279 #endif
01280 double clim_oh(
01281     double t,
01282     double lat,
01283     double p);
01284
01286 #ifdef _OPENACC
01287 #pragma acc routine (clim_tropo)
01288 #endif
01289 double clim_tropo(
01290     double t,
01291     double lat);
01292
01294 void day2doy(
01295     int year,
01296     int mon,
01297     int day,
01298     int *doy);
01299
01301 void doy2day(

```

```

01302     int year,
01303     int doy,
01304     int *mon,
01305     int *day);
01306
01307 void geo2cart(
01308     double z,
01309     double lon,
01310     double lat,
01311     double *x);
01312
01313 void get_met(
01314     ctl_t * ctl,
01315     double t,
01316     met_t ** met0,
01317     met_t ** met1);
01318
01319 void get_met_help(
01320     double t,
01321     int direct,
01322     char *metbase,
01323     double dt_met,
01324     char *filename);
01325
01326 void get_met_replace(
01327     char *orig,
01328     char *search,
01329     char *repl);
01330
01331 #ifdef _OPENACC
01332 #pragma acc routine (intpol_met_space_3d)
01333 #endif
01334 void intpol_met_space_3d(
01335     met_t * met,
01336     float array[EX][EY][EP],
01337     double p,
01338     double lon,
01339     double lat,
01340     double *var,
01341     int *ci,
01342     double *cw,
01343     int init);
01344
01345 #ifdef _OPENACC
01346 #pragma acc routine (intpol_met_space_2d)
01347 #endif
01348 void intpol_met_space_2d(
01349     met_t * met,
01350     float array[EX][EY],
01351     double lon,
01352     double lat,
01353     double *var,
01354     int *ci,
01355     double *cw,
01356     int init);
01357
01358 #ifdef _OPENACC
01359 #pragma acc routine (intpol_met_time_3d)
01360 #endif
01361 void intpol_met_time_3d(
01362     met_t * met0,
01363     float array0[EX][EY][EP],
01364     met_t * met1,
01365     float array1[EX][EY][EP],
01366     double ts,
01367     double p,
01368     double lon,
01369     double lat,
01370     double *var,
01371     int *ci,
01372     double *cw,
01373     int init);
01374
01375 #ifdef _OPENACC
01376 #pragma acc routine (intpol_met_time_2d)
01377 #endif
01378 void intpol_met_time_2d(
01379     met_t * met0,
01380     float array0[EX][EY],
01381     met_t * met1,
01382     float array1[EX][EY],
01383     double ts,
01384     double lon,
01385     double lat,
01386     double *var,
01387     int *ci,
01388     double *cw,
01389     int init);

```

```
01397     int init);
01398
01400 void jsec2time(
01401     double jsec,
01402     int *year,
01403     int *mon,
01404     int *day,
01405     int *hour,
01406     int *min,
01407     int *sec,
01408     double *remain);
01409
01411 #ifdef _OPENACC
01412 #pragma acc routine (lapse_rate)
01413 #endif
01414 double lapse_rate(
01415     double t,
01416     double h2o);
01417
01419 #ifdef _OPENACC
01420 #pragma acc routine (locate_irr)
01421 #endif
01422 int locate_irr(
01423     double *xx,
01424     int n,
01425     double x);
01426
01428 #ifdef _OPENACC
01429 #pragma acc routine (locate_reg)
01430 #endif
01431 int locate_reg(
01432     double *xx,
01433     int n,
01434     double x);
01435
01437 #ifdef _OPENACC
01438 #pragma acc routine (nat_temperature)
01439 #endif
01440 double nat_temperature(
01441     double p,
01442     double h2o,
01443     double hno3);
01444
01446 int read_atm(
01447     const char *filename,
01448     ctl_t * ctl,
01449     atm_t * atm);
01450
01452 void read_ctl(
01453     const char *filename,
01454     int argc,
01455     char *argv[],
01456     ctl_t * ctl);
01457
01459 int read_met(
01460     ctl_t * ctl,
01461     char *filename,
01462     met_t * met);
01463
01465 void read_met_cape(
01466     met_t * met);
01467
01469 void read_met_cloud(
01470     met_t * met);
01471
01473 void read_met_detrend(
01474     ctl_t * ctl,
01475     met_t * met);
01476
01478 void read_met_extrapolate(
01479     met_t * met);
01480
01482 void read_met_geopot(
01483     ctl_t * ctl,
01484     met_t * met);
01485
01487 void read_met_grid(
01488     char *filename,
01489     int ncid,
01490     ctl_t * ctl,
01491     met_t * met);
01492
01494 int read_met_help_3d(
01495     int ncid,
01496     char *varname,
01497     char *varname2,
01498     met_t * met,
```



```
01499     float dest[EX][EY][EP],
01500     float scl,
01501     int init);
01502
01504 int read_met_help_2d(
01505     int ncid,
01506     char *varname,
01507     char *varname2,
01508     met_t * met,
01509     float dest[EX][EY],
01510     float scl,
01511     int init);
01512
01514 void read_met_levels(
01515     int ncid,
01516     ctl_t * ctl,
01517     met_t * met);
01518
01520 void read_met_ml2pl(
01521     ctl_t * ctl,
01522     met_t * met,
01523     float var[EX][EY][EP]);
01524
01526 void read_met_pbl(
01527     met_t * met);
01528
01530 void read_met_periodic(
01531     met_t * met);
01532
01534 void read_met_pv(
01535     met_t * met);
01536
01538 void read_met_sample(
01539     ctl_t * ctl,
01540     met_t * met);
01541
01543 void read_met_surface(
01544     int ncid,
01545     met_t * met);
01546
01548 void read_met_tropo(
01549     ctl_t * ctl,
01550     met_t * met);
01551
01553 double scan_ctl(
01554     const char *filename,
01555     int argc,
01556     char *argv[],
01557     const char *varname,
01558     int arridx,
01559     const char *defvalue,
01560     char *value);
01561
01563 #ifdef _OPENACC
01564 #pragma acc routine (sedi)
01565 #endif
01566 double sedi(
01567     double p,
01568     double T,
01569     double r_p,
01570     double rho_p);
01571
01573 void spline(
01574     double *x,
01575     double *y,
01576     int n,
01577     double *x2,
01578     double *y2,
01579     int n2,
01580     int method);
01581
01583 #ifdef _OPENACC
01584 #pragma acc routine (stddev)
01585 #endif
01586 float stddev(
01587     float *data,
01588     int n);
01589
01591 void time2jsec(
01592     int year,
01593     int mon,
01594     int day,
01595     int hour,
01596     int min,
01597     int sec,
01598     double remain,
01599     double *jsec);
```

```

01600
01602 void timer(
01603     const char *name,
01604     const char *group,
01605     int output);
01606
01608 #ifdef _OPENACC
01609 #pragma acc routine (tropo_weight)
01610 #endif
01611 double tropo_weight(
01612     double t,
01613     double lat,
01614     double p);
01615
01617 void write_atm(
01618     const char *filename,
01619     ctl_t * ctl,
01620     atm_t * atm,
01621     double t);
01622
01624 void write_csi(
01625     const char *filename,
01626     ctl_t * ctl,
01627     atm_t * atm,
01628     double t);
01629
01631 void write_ens(
01632     const char *filename,
01633     ctl_t * ctl,
01634     atm_t * atm,
01635     double t);
01636
01638 void write_grid(
01639     const char *filename,
01640     ctl_t * ctl,
01641     met_t * met0,
01642     met_t * met1,
01643     atm_t * atm,
01644     double t);
01645
01647 void write_prof(
01648     const char *filename,
01649     ctl_t * ctl,
01650     met_t * met0,
01651     met_t * met1,
01652     atm_t * atm,
01653     double t);
01654
01656 void write_sample(
01657     const char *filename,
01658     ctl_t * ctl,
01659     met_t * met0,
01660     met_t * met1,
01661     atm_t * atm,
01662     double t);
01663
01665 void write_station(
01666     const char *filename,
01667     ctl_t * ctl,
01668     atm_t * atm,
01669     double t);
01670
01671 #endif /* LIBTRAC_H */

```

5.23 met_lapse.c File Reference

```
#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[])

5.23.1 Detailed Description

Calculate lapse rate statistics.

Definition in file [met_lapse.c](#).

5.23.2 Macro Definition Documentation

5.23.2.1 LAPSEMIN `#define LAPSEMIN -20.0`

Lapse rate minimum [K/km].

Definition at line 32 of file [met_lapse.c](#).

5.23.2.2 DLAPSE `#define DLAPSE 0.1`

Lapse rate bin size [K/km].

Definition at line 35 of file [met_lapse.c](#).

5.23.2.3 IDXMAX `#define IDXMAX 400`

Maximum number of histogram bins.

Definition at line 38 of file [met_lapse.c](#).

5.23.3 Function Documentation

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

Definition at line 44 of file [met_lapse.c](#).

```

00046     {
00047
00048         ctl_t ctl;
00049
00050         met_t *met;
00051
00052         FILE *out;
00053
00054         static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00055             z_mean;
00056
00057         static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00058             nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00059
00060         /* Allocate... */
00061         ALLOC(met, met_t, 1);
00062
00063         /* Check arguments... */
00064         if (argc < 4)
00065             ERRMSG("Give parameters: <ctl> <hist.tab> <met0> [ <met1> ... ]");
00066
00067         /* Read control parameters... */
00068         read_ctl(argv[1], argc, argv, &ctl);
00069         int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00070         double lat0 =
00071             (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00072         double lat1 =
00073             (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00074         double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00075         double z1 =
00076             (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00077         int intpol =
00078             (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);
00079
00080         /* Loop over files... */
00081         for (int i = 3; i < argc; i++) {
00082
00083             /* Read meteorological data... */
00084             if (!read_met(&ctl, argv[i], met))
00085                 continue;
00086
00087             /* Get altitude and pressure profiles... */
00088             for (int iz = 0; iz < met->np; iz++)
00089                 z[iz] = Z(met->p[iz]);
00090             for (int iz = 0; iz <= 250; iz++) {
00091                 z2[iz] = 0.0 + 0.1 * iz;
00092                 p2[iz] = P(z2[iz]);
00093             }
00094
00095             /* Loop over grid points... */
00096             for (int ix = 0; ix < met->nx; ix++)
00097                 for (int iy = 0; iy < met->ny; iy++) {
00098
00099                     /* Check latitude range... */
00100                     if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
00101                         continue;
00102
00103                     /* Interpolate temperature profile... */
00104                     for (int iz = 0; iz < met->np; iz++)
00105                         t[iz] = met->t[ix][iy][iz];
00106                     if (intpol == 1)
00107                         spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
00108                     else
00109                         for (int iz = 0; iz <= 250; iz++) {
00110                             int idx = locate_irr(z, met->np, z2[iz]);
00111                             t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
00112                         }
00113
00114                     /* Loop over vertical levels... */
00115                     for (int iz = 0; iz <= 250; iz++) {
00116
00117                         /* Check height range... */
00118                         if (z2[iz] < z0 || z2[iz] > z1)
00119                             continue;
00120
00121                         /* Check surface pressure... */
00122                         if (p2[iz] > met->ps[ix][iy])
00123                             continue;
00124
00125                         /* Get mean latitude and height... */

```

```

00126     lat_mean += met->lat[iy];
00127     z_mean += z2[iz];
00128     np++;
00129
00130     /* Get lapse rates within a vertical layer... */
00131     int nlapse = 0;
00132     double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00133         0, lapse_sig = 0;
00134     for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00135         lapse_max =
00136             GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00137         lapse_min =
00138             GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00139         lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00140         lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00141         nlapse++;
00142     }
00143     lapse_mean /= nlapse;
00144     lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00145
00146     /* Get histograms... */
00147     int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00148     if (idx >= 0 && idx < IDXMAX) {
00149         hist_max[idx]++;
00150         nhist_max++;
00151     }
00152
00153     idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00154     if (idx >= 0 && idx < IDXMAX) {
00155         hist_min[idx]++;
00156         nhist_min++;
00157     }
00158
00159     idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00160     if (idx >= 0 && idx < IDXMAX) {
00161         hist_mean[idx]++;
00162         nhist_mean++;
00163     }
00164
00165     idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00166     if (idx >= 0 && idx < IDXMAX) {
00167         hist_sig[idx]++;
00168         nhist_sig++;
00169     }
00170 }
00171 }
00172 }
00173
00174 /* Create output file... */
00175 LOG(1, "Write lapse rate data: %s", argv[2]);
00176 if (!(out = fopen(argv[2], "w")))
00177     ERRMSG("Cannot create file!");
00178
00179 /* Write header... */
00180 fprintf(out,
00181     "# $1 = mean altitude [km]\n"
00182     "# $2 = mean latitude [deg]\n"
00183     "# $3 = lapse rate [K/km]\n"
00184     "# $4 = counts of maxima per bin\n"
00185     "# $5 = total number of maxima\n"
00186     "# $6 = normalized frequency of maxima\n"
00187     "# $7 = counts of minima per bin\n"
00188     "# $8 = total number of minima\n"
00189     "# $9 = normalized frequency of minima\n"
00190     "# $10 = counts of means per bin\n"
00191     "# $11 = total number of means\n"
00192     "# $12 = normalized frequency of means\n"
00193     "# $13 = counts of sigmas per bin\n"
00194     "# $14 = total number of sigmas\n"
00195     "# $15 = normalized frequency of sigmas\n\n");
00196
00197 /* Write data... */
00198 double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00199 for (int idx = 0; idx < IDXMAX; idx++) {
00200     nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00201     nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00202     nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00203     nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00204 }
00205 for (int idx = 0; idx < IDXMAX; idx++)
00206     fprintf(out,
00207         "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
00208         z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00209         hist_max[idx], nhist_max,
00210         (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00211         nhist_min, (double) hist_min[idx] / (double) nmax_min,
00212         hist_mean[idx], nhist_mean,

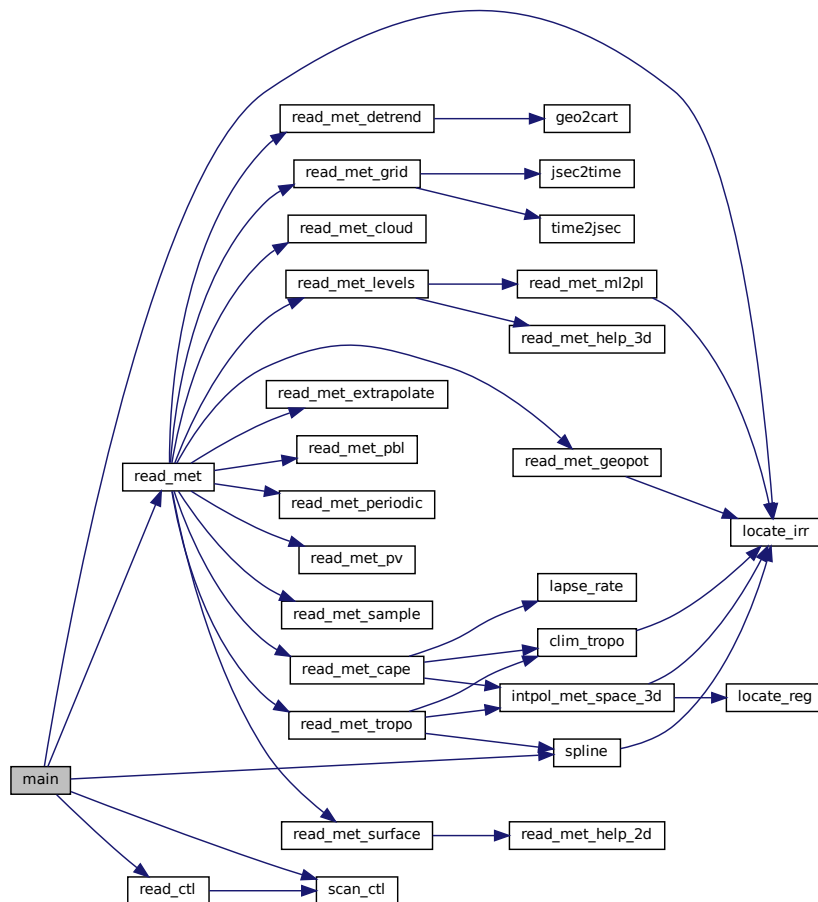
```

```

00213         (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00214         nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00215
00216     /* Close file... */
00217     fclose(out);
00218
00219     /* Free... */
00220     free(met);
00221
00222     return EXIT_SUCCESS;
00223 }

```

Here is the call graph for this function:



5.24 met_lapse.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) 2013-2021 Forschungszentrum Juelich GmbH

```

```

00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028     Dimensions...
00029 ----- */
00030
00032 #define LAPSEMIN -20.0
00033
00035 #define DLAPSE 0.1
00036
00038 #define IDXMAX 400
00039
00040 /* -----
00041     Main...
00042 ----- */
00043
00044 int main(
00045     int argc,
00046     char *argv[]) {
00047
00048     ctl_t ctl;
00049
00050     met_t *met;
00051
00052     FILE *out;
00053
00054     static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00055                 z_mean;
00056
00057     static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00058              nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <hist.tab> <met0> [ <met1> ... ]");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00070     double lat0 =
00071         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00072     double lat1 =
00073         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00074     double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00075     double z1 =
00076         (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00077     int intpol =
00078         (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);
00079
00080     /* Loop over files... */
00081     for (int i = 3; i < argc; i++) {
00082
00083         /* Read meteorological data... */
00084         if (!read_met(&ctl, argv[i], met))
00085             continue;
00086
00087         /* Get altitude and pressure profiles... */
00088         for (int iz = 0; iz < met->np; iz++)
00089             z[iz] = Z(met->p[iz]);
00090         for (int iz = 0; iz <= 250; iz++) {
00091             z2[iz] = 0.0 + 0.1 * iz;
00092             p2[iz] = P(z2[iz]);
00093         }
00094
00095         /* Loop over grid points... */
00096         for (int ix = 0; ix < met->nx; ix++)
00097             for (int iy = 0; iy < met->ny; iy++) {
00098
00099                 /* Check latitude range... */
00100                 if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
00101                     continue;
00102
00103                 /* Interpolate temperature profile... */
00104                 for (int iz = 0; iz < met->np; iz++)
00105                     t[iz] = met->t[ix][iy][iz];
00106                 if (intpol == 1)
00107                     spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
00108                 else
00109                     for (int iz = 0; iz <= 250; iz++) {
00110                         int idx = locate_irr(z, met->np, z2[iz]);
00111                         t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
00112                     }

```

```

00113
00114     /* Loop over vertical levels... */
00115     for (int iz = 0; iz <= 250; iz++) {
00116
00117         /* Check height range... */
00118         if (z2[iz] < z0 || z2[iz] > z1)
00119             continue;
00120
00121         /* Check surface pressure... */
00122         if (p2[iz] > met->ps[ix][iy])
00123             continue;
00124
00125         /* Get mean latitude and height... */
00126         lat_mean += met->lat[iy];
00127         z_mean += z2[iz];
00128         np++;
00129
00130         /* Get lapse rates within a vertical layer... */
00131         int nlapse = 0;
00132         double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00133             0, lapse_sig = 0;
00134         for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00135             lapse_max =
00136                 GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00137             lapse_min =
00138                 GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00139             lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00140             lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00141             nlapse++;
00142         }
00143         lapse_mean /= nlapse;
00144         lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00145
00146         /* Get histograms... */
00147         int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00148         if (idx >= 0 && idx < IDXMAX) {
00149             hist_max[idx]++;
00150             nhist_max++;
00151         }
00152
00153         idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00154         if (idx >= 0 && idx < IDXMAX) {
00155             hist_min[idx]++;
00156             nhist_min++;
00157         }
00158
00159         idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00160         if (idx >= 0 && idx < IDXMAX) {
00161             hist_mean[idx]++;
00162             nhist_mean++;
00163         }
00164
00165         idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00166         if (idx >= 0 && idx < IDXMAX) {
00167             hist_sig[idx]++;
00168             nhist_sig++;
00169         }
00170     }
00171 }
00172 }
00173
00174 /* Create output file... */
00175 LOG(1, "Write lapse rate data: %s", argv[2]);
00176 if (!(out = fopen(argv[2], "w")))
00177     ERRMSG("Cannot create file!");
00178
00179 /* Write header... */
00180 fprintf(out,
00181     "# $1 = mean altitude [km]\n"
00182     "# $2 = mean latitude [deg]\n"
00183     "# $3 = lapse rate [K/km]\n"
00184     "# $4 = counts of maxima per bin\n"
00185     "# $5 = total number of maxima\n"
00186     "# $6 = normalized frequency of maxima\n"
00187     "# $7 = counts of minima per bin\n"
00188     "# $8 = total number of minima\n"
00189     "# $9 = normalized frequency of minima\n"
00190     "# $10 = counts of means per bin\n"
00191     "# $11 = total number of means\n"
00192     "# $12 = normalized frequency of means\n"
00193     "# $13 = counts of sigmas per bin\n"
00194     "# $14 = total number of sigmas\n"
00195     "# $15 = normalized frequency of sigmas\n\n");
00196
00197 /* Write data... */
00198 double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00199 for (int idx = 0; idx < IDXMAX; idx++) {

```



```

00200     nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00201     nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00202     nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00203     nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00204 }
00205 for (int idx = 0; idx < IDXMAX; idx++)
00206     fprintf(out,
00207         "%g %g %g %d %d %g %d %d %g %d %d %g\n",
00208         z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00209         hist_max[idx], nhist_max,
00210         (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00211         nhist_min, (double) hist_min[idx] / (double) nmax_min,
00212         hist_mean[idx], nhist_mean,
00213         (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00214         nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00215
00216 /* Close file... */
00217 fclose(out);
00218
00219 /* Free... */
00220 free(met);
00221
00222 return EXIT_SUCCESS;
00223 }

```

5.25 met_map.c File Reference

```
#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[])`

5.25.1 Detailed Description

Extract map from meteorological data.

Definition in file [met_map.c](#).

5.25.2 Macro Definition Documentation

5.25.2.1 NX `#define NX 1441`

Maximum number of longitudes.

Definition at line 32 of file [met_map.c](#).

5.25.2.2 NY #define NY 721

Maximum number of latitudes.

Definition at line 35 of file [met_map.c](#).

5.25.3 Function Documentation

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

Definition at line 41 of file [met_map.c](#).

```
00043     {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
00052         zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
00053         ptm[NX][NY], t, pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY],
00054         w, wm[NX][NY], h2o, h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00055         hno3m[NX][NY], ohm[NX][NY], tdewm[NX][NY], ticem[NX][NY], tnatm[NX][NY],
00056         lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv, pvm[NX][NY],
00057         zt, ztm[NX][NY], tt, ttm[NX][NY], pct, pctm[NX][NY], pcb, pcbm[NX][NY],
00058         cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
00059         pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
00060         rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
00061         lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00062
00063     static int i, ix, iy, np[NX][NY], npc[NX][NY], npt[NX][NY], nx, ny, ci[3];
00064
00065     /* Allocate... */
00066     ALLOC(met, met_t, 1);
00067
00068     /* Check arguments... */
00069     if (argc < 4)
00070         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00071
00072     /* Read control parameters... */
00073     read_ctl(argv[1], argc, argv, &ctl);
00074     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00075     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00076     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00077     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00078     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00079     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00080     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00081     theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00082
00083     /* Loop over files... */
00084     for (i = 3; i < argc; i++) {
00085
00086         /* Read meteorological data... */
00087         if (!read_met(&ctl, argv[i], met))
00088             continue;
00089
00090         /* Set horizontal grid... */
00091         if (dlon <= 0)
00092             dlon = fabs(met->lon[1] - met->lon[0]);
00093         if (dlat <= 0)
00094             dlat = fabs(met->lat[1] - met->lat[0]);
00095         if (lon0 < -360 && lon1 > 360) {
00096             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00097             lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00098         }
00099         nx = ny = 0;
00100         for (lon = lon0; lon <= lon1; lon += dlon) {
00101             lons[nx] = lon;
00102             if (++nx > NX)
00103                 ERRMSG("Too many longitudes!");
```

```

00104     }
00105     if (lat0 < -90 && lat1 > 90) {
00106         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00107         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00108     }
00109     for (lat = lat0; lat <= lat1; lat += dlat) {
00110         lats[ny] = lat;
00111         if ((++ny) > NY)
00112             ERRMSG("Too many latitudes!");
00113     }
00114
00115     /* Average... */
00116     for (ix = 0; ix < nx; ix++)
00117         for (iy = 0; iy < ny; iy++) {
00118
00119             /* Find pressure level for given theta level... */
00120             if (theta > 0) {
00121                 ptop = met->p[met->np - 1];
00122                 pbot = met->p[0];
00123                 do {
00124                     p0 = 0.5 * (ptop + pbot);
00125                     intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00126                                         &t0, ci, cw, 1);
00127                     if (THETA(p0, t0) > theta)
00128                         ptop = p0;
00129                     else
00130                         pbot = p0;
00131                 } while (fabs(ptop - pbot) > 1e-5);
00132             }
00133
00134             /* Interpolate meteo data... */
00135             INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00136
00137             /* Averaging... */
00138             timem[ix][iy] += met->time;
00139             zm[ix][iy] += z;
00140             pm[ix][iy] += p0;
00141             tm[ix][iy] += t;
00142             um[ix][iy] += u;
00143             vm[ix][iy] += v;
00144             wm[ix][iy] += w;
00145             pvm[ix][iy] += pv;
00146             h2om[ix][iy] += h2o;
00147             o3m[ix][iy] += o3;
00148             lwcm[ix][iy] += lwc;
00149             iwcm[ix][iy] += iwc;
00150             psm[ix][iy] += ps;
00151             tsm[ix][iy] += ts;
00152             zsm[ix][iy] += zs;
00153             usm[ix][iy] += us;
00154             vsm[ix][iy] += vs;
00155             pblm[ix][iy] += pbl;
00156             pctm[ix][iy] += pct;
00157             pcbm[ix][iy] += pcb;
00158             clm[ix][iy] += cl;
00159             if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00160                 && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00161                 plclm[ix][iy] += plcl;
00162                 plfcm[ix][iy] += plfc;
00163                 pelm[ix][iy] += pel;
00164                 capem[ix][iy] += cape;
00165                 cinm[ix][iy] += cin;
00166                 npc[ix][iy]++;
00167             }
00168             if (gsl_finite(pt)) {
00169                 ptm[ix][iy] += pt;
00170                 ztm[ix][iy] += zt;
00171                 ttm[ix][iy] += tt;
00172                 h2otm[ix][iy] += h2ot;
00173                 npt[ix][iy]++;
00174             }
00175             hno3m[ix][iy] += clim_hno3(met->time, lats[iy], p0);
00176             tnatm[ix][iy] +=
00177                 nat_temperature(p0, h2o, clim_hno3(met->time, lats[iy], p0));
00178             ohm[ix][iy] += clim_oh(met->time, lats[iy], p0);
00179             rhm[ix][iy] += RH(p0, t, h2o);
00180             rhicem[ix][iy] += RHICE(p0, t, h2o);
00181             tdewm[ix][iy] += TDEW(p0, h2o);
00182             ticem[ix][iy] += TICE(p0, h2o);
00183             np[ix][iy]++;
00184         }
00185     }
00186
00187     /* Create output file... */
00188     LOG(1, "Write meteorological data file: %s", argv[2]);
00189     if (!(out = fopen(argv[2], "w")))
00190         ERRMSG("Cannot create file!");

```

```

00191
00192 /* Write header... */
00193 fprintf(out,
00194     "# $1 = time [s]\n"
00195     "# $2 = altitude [km]\n"
00196     "# $3 = longitude [deg]\n"
00197     "# $4 = latitude [deg]\n"
00198     "# $5 = pressure [hPa]\n"
00199     "# $6 = temperature [K]\n"
00200     "# $7 = zonal wind [m/s]\n"
00201     "# $8 = meridional wind [m/s]\n"
00202     "# $9 = vertical velocity [hPa/s]\n"
00203     "# $10 = H2O volume mixing ratio [ppv]\n");
00204 fprintf(out,
00205     "# $11 = O3 volume mixing ratio [ppv]\n"
00206     "# $12 = geopotential height [km]\n"
00207     "# $13 = potential vorticity [PVU]\n"
00208     "# $14 = surface pressure [hPa]\n"
00209     "# $15 = surface temperature [K]\n"
00210     "# $16 = surface geopotential height [km]\n"
00211     "# $17 = surface zonal wind [m/s]\n"
00212     "# $18 = surface meridional wind [m/s]\n"
00213     "# $19 = tropopause pressure [hPa]\n"
00214     "# $20 = tropopause geopotential height [km]\n");
00215 fprintf(out,
00216     "# $21 = tropopause temperature [K]\n"
00217     "# $22 = tropopause water vapor [ppv]\n"
00218     "# $23 = cloud liquid water content [kg/kg]\n"
00219     "# $24 = cloud ice water content [kg/kg]\n"
00220     "# $25 = total column cloud water [kg/m^2]\n"
00221     "# $26 = cloud top pressure [hPa]\n"
00222     "# $27 = cloud bottom pressure [hPa]\n"
00223     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00224     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00225     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00226 fprintf(out,
00227     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00228     "# $32 = convective inhibition (CIN) [J/kg]\n"
00229     "# $33 = relative humidity over water [%]\n"
00230     "# $34 = relative humidity over ice [%]\n"
00231     "# $35 = dew point temperature [K]\n"
00232     "# $36 = frost point temperature [K]\n"
00233     "# $37 = NAT temperature [K]\n"
00234     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00235     "# $39 = OH concentration [molec/cm^3]\n"
00236     "# $40 = boundary layer pressure [hPa]\n");
00237 fprintf(out,
00238     "# $41 = number of data points\n"
00239     "# $42 = number of tropopause data points\n"
00240     "# $43 = number of CAPE data points\n");
00241
00242 /* Write data... */
00243 for (iy = 0; iy < ny; iy++) {
00244     fprintf(out, "\n");
00245     for (ix = 0; ix < nx; ix++)
00246         fprintf(out,
00247             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n"
00248             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00249             " %g %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
00250             timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00251             lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00252             tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00253             vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00254             h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00255             zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00256             psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00257             zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00258             vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
00259             ztm[ix][iy] / npt[ix][iy], ttm[ix][iy] / npt[ix][iy],
00260             h2otm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
00261             iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00262             pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
00263             plclm[ix][iy] / npc[ix][iy], plfcm[ix][iy] / npc[ix][iy],
00264             pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
00265             cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
00266             rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
00267             ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
00268             hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
00269             pblm[ix][iy] / np[ix][iy], np[ix][iy],
00270             npt[ix][iy], npc[ix][iy]);
00271     }
00272
00273 /* Close file... */
00274 fclose(out);
00275
00276 /* Free... */
00277 free(met);

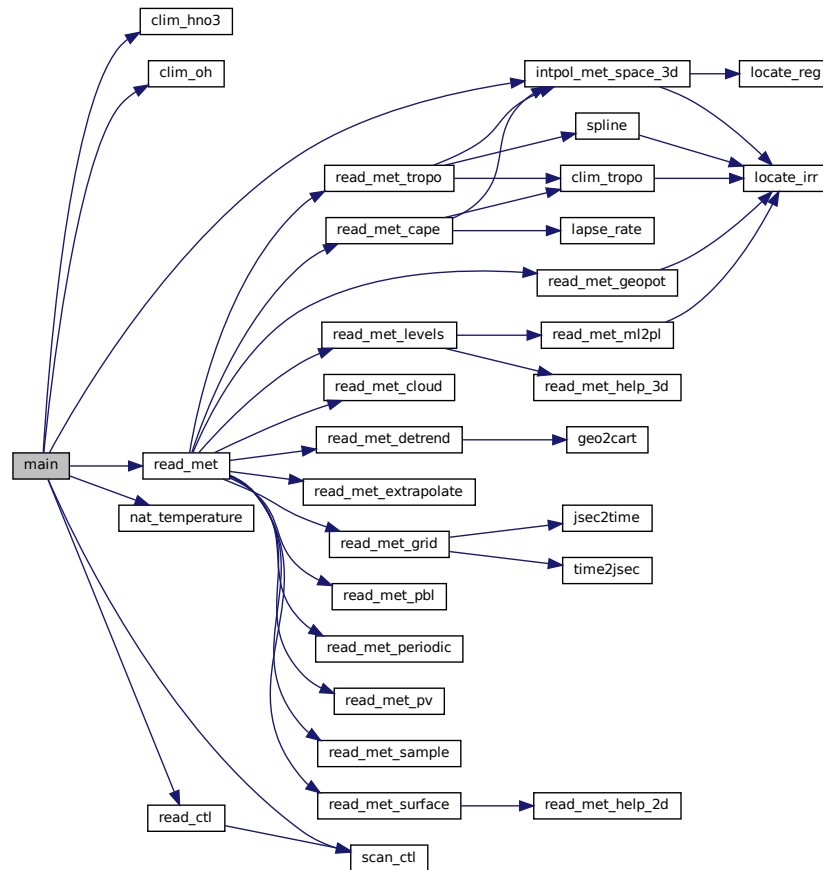
```

```

00278
00279     return EXIT_SUCCESS;
00280 }

```

Here is the call graph for this function:



5.26 met_map.c

```

00001 /*
00002     This file is part of MPTRAC.
00003
00004     MPTRAC is free software: you can redistribute it and/or modify
00005     it under the terms of the GNU General Public License as published by
00006     the Free Software Foundation, either version 3 of the License, or
00007     (at your option) any later version.
00008
00009     MPTRAC is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU General Public License for more details.
00013
00014     You should have received a copy of the GNU General Public License
00015     along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017     Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028     Dimensions...
00029     ----- */
00030
00032 #define NX 1441

```

```

00033
00035 #define NY 721
00036
00037 /* -----
00038     Main...
00039     ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
00052         zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
00053         ptm[NX][NY], t, pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY],
00054         w, wm[NX][NY], h2o, h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00055         hno3m[NX][NY], ohm[NX][NY], tdewm[NX][NY], ticem[NX][NY], tnatm[NX][NY],
00056         lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv, pvm[NX][NY],
00057         zt, ztm[NX][NY], tt, ttm[NX][NY], pct, pctm[NX][NY], pcb, pcbm[NX][NY],
00058         cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
00059         pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
00060         rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
00061         lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00062
00063     static int i, ix, iy, np[NX][NY], npc[NX][NY], npt[NX][NY], nx, ny, ci[3];
00064
00065     /* Allocate... */
00066     ALLOC(met, met_t, 1);
00067
00068     /* Check arguments... */
00069     if (argc < 4)
00070         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00071
00072     /* Read control parameters... */
00073     read_ctl(argv[1], argc, argv, &ctl);
00074     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00075     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00076     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00077     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00078     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00079     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00080     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00081     theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00082
00083     /* Loop over files... */
00084     for (i = 3; i < argc; i++) {
00085
00086         /* Read meteorological data... */
00087         if (!read_met(&ctl, argv[i], met))
00088             continue;
00089
00090         /* Set horizontal grid... */
00091         if (dlon <= 0)
00092             dlon = fabs(met->lon[1] - met->lon[0]);
00093         if (dlat <= 0)
00094             dlat = fabs(met->lat[1] - met->lat[0]);
00095         if (lon0 < -360 && lon1 > 360) {
00096             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00097             lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00098         }
00099         nx = ny = 0;
00100         for (lon = lon0; lon <= lon1; lon += dlon) {
00101             lons[nx] = lon;
00102             if (++nx > NX)
00103                 ERRMSG("Too many longitudes!");
00104         }
00105         if (lat0 < -90 && lat1 > 90) {
00106             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00107             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00108         }
00109         for (lat = lat0; lat <= lat1; lat += dlat) {
00110             lats[ny] = lat;
00111             if (++ny > NY)
00112                 ERRMSG("Too many latitudes!");
00113         }
00114
00115         /* Average... */
00116         for (ix = 0; ix < nx; ix++)
00117             for (iy = 0; iy < ny; iy++) {
00118
00119                 /* Find pressure level for given theta level... */
00120                 if (theta > 0) {

```

```

00121     ptop = met->p[met->np - 1];
00122     pbot = met->p[0];
00123     do {
00124         p0 = 0.5 * (ptop + pbot);
00125         interpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00126                               &t0, ci, cw, 1);
00127         if (THETA(p0, t0) > theta)
00128             ptop = p0;
00129         else
00130             pbot = p0;
00131     } while (fabs(ptop - pbot) > 1e-5);
00132 }
00133
00134 /* Interpolate meteo data... */
00135 INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00136
00137 /* Averaging... */
00138 timem[ix][iy] += met->time;
00139 zm[ix][iy] += z;
00140 pm[ix][iy] += p0;
00141 tm[ix][iy] += t;
00142 um[ix][iy] += u;
00143 vm[ix][iy] += v;
00144 wm[ix][iy] += w;
00145 pvm[ix][iy] += pv;
00146 h2om[ix][iy] += h2o;
00147 o3m[ix][iy] += o3;
00148 lwcm[ix][iy] += lwc;
00149 iwcm[ix][iy] += iwc;
00150 psm[ix][iy] += ps;
00151 tsm[ix][iy] += ts;
00152 zsm[ix][iy] += zs;
00153 usm[ix][iy] += us;
00154 vsm[ix][iy] += vs;
00155 pblm[ix][iy] += pbl;
00156 pctm[ix][iy] += pct;
00157 pcbm[ix][iy] += pcb;
00158 clm[ix][iy] += cl;
00159 if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00160     && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00161     plclm[ix][iy] += plcl;
00162     plfcm[ix][iy] += plfc;
00163     pelm[ix][iy] += pel;
00164     capem[ix][iy] += cape;
00165     cinm[ix][iy] += cin;
00166     npc[ix][iy]++;
00167 }
00168 if (gsl_finite(pt)) {
00169     ptm[ix][iy] += pt;
00170     ztm[ix][iy] += zt;
00171     ttm[ix][iy] += tt;
00172     h2otm[ix][iy] += h2ot;
00173     npt[ix][iy]++;
00174 }
00175 hno3m[ix][iy] += clim_hno3(met->time, lats[iy], p0);
00176 tnatm[ix][iy] +=
00177     nat_temperature(p0, h2o, clim_hno3(met->time, lats[iy], p0));
00178 ohm[ix][iy] += clim_oh(met->time, lats[iy], p0);
00179 rhm[ix][iy] += RH(p0, t, h2o);
00180 rhicem[ix][iy] += RHICE(p0, t, h2o);
00181 tdewm[ix][iy] += TDEW(p0, h2o);
00182 ticem[ix][iy] += TICE(p0, h2o);
00183 np[ix][iy]++;
00184 }
00185 }
00186
00187 /* Create output file... */
00188 LOG(1, "Write meteorological data file: %s", argv[2]);
00189 if (!(out = fopen(argv[2], "w")))
00190     ERRMSG("Cannot create file!");
00191
00192 /* Write header... */
00193 fprintf(out,
00194         "# $1 = time [s]\n"
00195         "# $2 = altitude [km]\n"
00196         "# $3 = longitude [deg]\n"
00197         "# $4 = latitude [deg]\n"
00198         "# $5 = pressure [hPa]\n"
00199         "# $6 = temperature [K]\n"
00200         "# $7 = zonal wind [m/s]\n"
00201         "# $8 = meridional wind [m/s]\n"
00202         "# $9 = vertical velocity [hPa/s]\n"
00203         "# $10 = H2O volume mixing ratio [ppv]\n");
00204 fprintf(out,
00205         "# $11 = O3 volume mixing ratio [ppv]\n"
00206         "# $12 = geopotential height [km]\n"
00207         "# $13 = potential vorticity [PVU]\n");

```

```

00208         "# $14 = surface pressure [hPa]\n"
00209         "# $15 = surface temperature [K]\n"
00210         "# $16 = surface geopotential height [km]\n"
00211         "# $17 = surface zonal wind [m/s]\n"
00212         "# $18 = surface meridional wind [m/s]\n"
00213         "# $19 = tropopause pressure [hPa]\n"
00214         "# $20 = tropopause geopotential height [km]\n");
00215 fprintf(out,
00216         "# $21 = tropopause temperature [K]\n"
00217         "# $22 = tropopause water vapor [ppv]\n"
00218         "# $23 = cloud liquid water content [kg/kg]\n"
00219         "# $24 = cloud ice water content [kg/kg]\n"
00220         "# $25 = total column cloud water [kg/m^2]\n"
00221         "# $26 = cloud top pressure [hPa]\n"
00222         "# $27 = cloud bottom pressure [hPa]\n"
00223         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00224         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00225         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00226 fprintf(out,
00227         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00228         "# $32 = convective inhibition (CIN) [J/kg]\n"
00229         "# $33 = relative humidity over water [%]\n"
00230         "# $34 = relative humidity over ice [%]\n"
00231         "# $35 = dew point temperature [K]\n"
00232         "# $36 = frost point temperature [K]\n"
00233         "# $37 = NAT temperature [K]\n"
00234         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00235         "# $39 = OH concentration [molec/cm^3]\n"
00236         "# $40 = boundary layer pressure [hPa]\n");
00237 fprintf(out,
00238         "# $41 = number of data points\n"
00239         "# $42 = number of tropopause data points\n"
00240         "# $43 = number of CAPE data points\n");
00241
00242 /* Write data... */
00243 for (iy = 0; iy < ny; iy++) {
00244     fprintf(out, "\n");
00245     for (ix = 0; ix < nx; ix++)
00246         fprintf(out,
00247             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n"
00248             " %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00249             " %g %g %g %g %g %g %g %g %g %d %d %d\n",
00250             timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00251             lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00252             tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00253             vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00254             h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00255             zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00256             psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00257             zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00258             vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
00259             ztm[ix][iy] / npt[ix][iy], ttm[ix][iy] / npt[ix][iy],
00260             h2otm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
00261             iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00262             pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
00263             plclm[ix][iy] / npc[ix][iy], plfcm[ix][iy] / npc[ix][iy],
00264             pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
00265             cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
00266             rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
00267             ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
00268             hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
00269             pblm[ix][iy] / np[ix][iy], np[ix][iy],
00270             npt[ix][iy], npc[ix][iy]);
00271     }
00272
00273 /* Close file... */
00274 fclose(out);
00275
00276 /* Free... */
00277 free(met);
00278
00279 return EXIT_SUCCESS;
00280 }

```

5.27 met_prof.c File Reference

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


Macros

- `#define NZ 1000`
Maximum number of altitudes.

Functions

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

5.27.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file [met_prof.c](#).

5.27.2 Macro Definition Documentation

5.27.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line 32 of file [met_prof.c](#).

5.27.3 Function Documentation

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

Definition at line 38 of file [met_prof.c](#).

```
00040     {
00041
00042     ctl_t ctl;
00043
00044     met_t *met;
00045
00046     FILE *out;
00047
00048     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050     wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00051     iwc, iwcmm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00052     vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
00053     cl, clm[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
00054     cape, capem[NZ], cin, cinm[NZ], tt, ttmm[NZ], zm[NZ], zt, ztm[NZ],
00055     pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
00056     tnatm[NZ], hno3m[NZ], ohm[NZ], cw[3];
00057
00058     static int i, iz, np[NZ], npc[NZ], npt[NZ], nz, ci[3];
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
```

```

00063  /* Check arguments... */
00064  if (argc < 4)
00065      ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00066
00067  /* Read control parameters... */
00068  read_ctl(argv[1], argc, argv, &ctl);
00069  z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00070  z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00071  dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00072  lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00073  lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00074  dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00075  lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00076  lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00077  dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00078
00079  /* Loop over input files... */
00080  for (i = 3; i < argc; i++) {
00081
00082      /* Read meteorological data... */
00083      if (!read_met(&ctl, argv[i], met))
00084          continue;
00085
00086      /* Set vertical grid... */
00087      if (z0 < 0)
00088          z0 = Z(met->p[0]);
00089      if (z1 < 0)
00090          z1 = Z(met->p[met->np - 1]);
00091      nz = 0;
00092      if (dz < 0) {
00093          for (iz = 0; iz < met->np; iz++)
00094              if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00095                  plev[nz] = met->p[iz];
00096                  if ((++nz) > NZ)
00097                      ERRMSG("Too many pressure levels!");
00098              }
00099      } else
00100          for (z = z0; z <= z1; z += dz) {
00101              plev[nz] = P(z);
00102              if ((++nz) > NZ)
00103                  ERRMSG("Too many pressure levels!");
00104          }
00105
00106      /* Set horizontal grid... */
00107      if (dlon <= 0)
00108          dlon = fabs(met->lon[1] - met->lon[0]);
00109      if (dlat <= 0)
00110          dlat = fabs(met->lat[1] - met->lat[0]);
00111
00112      /* Average... */
00113      for (iz = 0; iz < nz; iz++)
00114          for (lon = lon0; lon <= lon1; lon += dlon)
00115              for (lat = lat0; lat <= lat1; lat += dlat) {
00116
00117                  /* Interpolate meteo data... */
00118                  INTPOL_SPACE_ALL(plev[iz], lon, lat);
00119
00120                  /* Averaging... */
00121                  if (gsl_finite(t) && gsl_finite(u)
00122                      && gsl_finite(v) && gsl_finite(w)) {
00123                      timem[iz] += met->time;
00124                      lonm[iz] += lon;
00125                      latm[iz] += lat;
00126                      zm[iz] += z;
00127                      tm[iz] += t;
00128                      um[iz] += u;
00129                      vm[iz] += v;
00130                      wm[iz] += w;
00131                      pvm[iz] += pv;
00132                      h2om[iz] += h2o;
00133                      o3m[iz] += o3;
00134                      lwcm[iz] += lwc;
00135                      iwcm[iz] += iwc;
00136                      psm[iz] += ps;
00137                      tsm[iz] += ts;
00138                      zsm[iz] += zs;
00139                      usm[iz] += us;
00140                      vsm[iz] += vs;
00141                      pblm[iz] += pbl;
00142                      pctm[iz] += pct;
00143                      pcbm[iz] += pcb;
00144                      clm[iz] += cl;
00145                      if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00146                          && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00147                          plclm[iz] += plcl;
00148                          plfcm[iz] += plfc;
00149                          pelm[iz] += pel;

```

```

00150         capem[iz] += cape;
00151         cinm[iz] += cin;
00152         npc[iz]++;
00153     }
00154     if (gsl_finite(pt)) {
00155         ptm[iz] += pt;
00156         ztm[iz] += zt;
00157         ttm[iz] += tt;
00158         h2otm[iz] += h2ot;
00159         npt[iz]++;
00160     }
00161     rhm[iz] += RH(plev[iz], t, h2o);
00162     rhicem[iz] += RHICE(plev[iz], t, h2o);
00163     tdewm[iz] += TDEW(plev[iz], h2o);
00164     ticem[iz] += TICE(plev[iz], h2o);
00165     hno3m[iz] += clim_hno3(met->time, lat, plev[iz]);
00166     tnatm[iz] +=
00167         nat_temperature(plev[iz], h2o,
00168             clim_hno3(met->time, lat, plev[iz]));
00169     ohm[iz] += clim_oh(met->time, lat, plev[iz]);
00170     np[iz]++;
00171 }
00172 }
00173 }
00174
00175 /* Create output file... */
00176 LOG(1, "Write meteorological data file: %s", argv[2]);
00177 if (!(out = fopen(argv[2], "w")))
00178     ERRMSG("Cannot create file!");
00179
00180 /* Write header... */
00181 fprintf(out,
00182     "# $1 = time [s]\n"
00183     "# $2 = altitude [km]\n"
00184     "# $3 = longitude [deg]\n"
00185     "# $4 = latitude [deg]\n"
00186     "# $5 = pressure [hPa]\n"
00187     "# $6 = temperature [K]\n"
00188     "# $7 = zonal wind [m/s]\n"
00189     "# $8 = meridional wind [m/s]\n"
00190     "# $9 = vertical velocity [hPa/s]\n"
00191     "# $10 = H2O volume mixing ratio [ppv]\n");
00192 fprintf(out,
00193     "# $11 = O3 volume mixing ratio [ppv]\n"
00194     "# $12 = geopotential height [km]\n"
00195     "# $13 = potential vorticity [PVU]\n"
00196     "# $14 = surface pressure [hPa]\n"
00197     "# $15 = surface temperature [K]\n"
00198     "# $16 = surface geopotential height [km]\n"
00199     "# $17 = surface zonal wind [m/s]\n"
00200     "# $18 = surface meridional wind [m/s]\n"
00201     "# $19 = tropopause pressure [hPa]\n"
00202     "# $20 = tropopause geopotential height [km]\n");
00203 fprintf(out,
00204     "# $21 = tropopause temperature [K]\n"
00205     "# $22 = tropopause water vapor [ppv]\n"
00206     "# $23 = cloud liquid water content [kg/kg]\n"
00207     "# $24 = cloud ice water content [kg/kg]\n"
00208     "# $25 = total column cloud water [kg/m^2]\n"
00209     "# $26 = cloud top pressure [hPa]\n"
00210     "# $27 = cloud bottom pressure [hPa]\n"
00211     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00212     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00213     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00214 fprintf(out,
00215     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00216     "# $32 = convective inhibition (CIN) [J/kg]\n"
00217     "# $33 = relative humidity over water [%]\n"
00218     "# $34 = relative humidity over ice [%]\n"
00219     "# $35 = dew point temperature [K]\n"
00220     "# $36 = frost point temperature [K]\n"
00221     "# $37 = NAT temperature [K]\n"
00222     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00223     "# $39 = OH concentration [molec/cm^3]\n"
00224     "# $40 = boundary layer pressure [hPa]\n");
00225 fprintf(out,
00226     "# $41 = number of data points\n"
00227     "# $42 = number of tropopause data points\n"
00228     "# $43 = number of CAPE data points\n");
00229
00230 /* Write data... */
00231 for (iz = 0; iz < nz; iz++)
00232     fprintf(out,
00233         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n"
00234         " %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00235         " %g %g %g %g %g %g %g %g %g %d %d\n",
00236         timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],

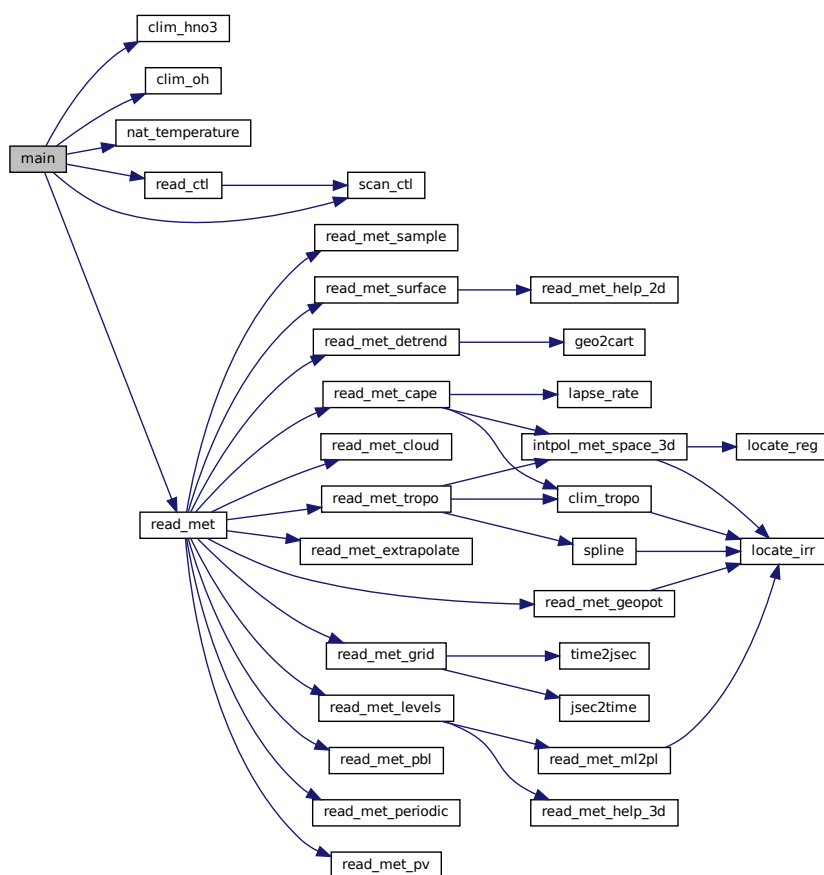
```

```

00237     latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00238     vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00239     o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00240     psm[iz] / np[iz], tsm[iz] / np[iz], zsm[iz] / np[iz],
00241     usm[iz] / np[iz], vsm[iz] / np[iz], ptm[iz] / npt[iz],
00242     ztm[iz] / npt[iz], ttm[iz] / npt[iz], h2otm[iz] / npt[iz],
00243     lwcm[iz] / np[iz], iwcm[iz] / np[iz], clm[iz] / np[iz],
00244     pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
00245     plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
00246     cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
00247     tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
00248     hno3m[iz] / np[iz], ohm[iz] / np[iz], pblm[iz] / np[iz],
00249     np[iz], npt[iz], npc[iz]);
00250
00251     /* Close file... */
00252     fclose(out);
00253
00254     /* Free... */
00255     free(met);
00256
00257     return EXIT_SUCCESS;
00258 }

```

Here is the call graph for this function:



5.28 met_prof.c

```

00001 /*
00002     This file is part of MPTRAC.
00003
00004     MPTRAC is free software: you can redistribute it and/or modify
00005     it under the terms of the GNU General Public License as published by
00006     the Free Software Foundation, either version 3 of the License, or
00007     (at your option) any later version.

```

```

00008
00009 MPTRAC is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU General Public License for more details.
00013
00014 You should have received a copy of the GNU General Public License
00015 along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028     Dimensions...
00029     ----- */
00030
00032 #define NZ 1000
00033
00034 /* -----
00035     Main...
00036     ----- */
00037
00038 int main(
00039     int argc,
00040     char *argv[]) {
00041
00042     ctl_t ctl;
00043
00044     met_t *met;
00045
00046     FILE *out;
00047
00048     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049         lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050         wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00051         iwc, iwcm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00052         vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
00053         cl, clm[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
00054         cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
00055         pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
00056         tnatm[NZ], hno3m[NZ], ohm[NZ], cw[3];
00057
00058     static int i, iz, np[NZ], npc[NZ], npt[NZ], nz, ci[3];
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00070     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00071     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00072     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00073     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00074     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00075     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00076     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00077     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00078
00079     /* Loop over input files... */
00080     for (i = 3; i < argc; i++) {
00081
00082         /* Read meteorological data... */
00083         if (!read_met(&ctl, argv[i], met))
00084             continue;
00085
00086         /* Set vertical grid... */
00087         if (z0 < 0)
00088             z0 = Z(met->p[0]);
00089         if (z1 < 0)
00090             z1 = Z(met->p[met->np - 1]);
00091         nz = 0;
00092         if (dz < 0) {
00093             for (iz = 0; iz < met->np; iz++)
00094                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00095                     plev[nz] = met->p[iz];
00096                     if (++nz > NZ)
00097                         ERRMSG("Too many pressure levels!");
00098                 }
00099             } else
00100                 for (z = z0; z <= z1; z += dz) {

```

```

00101     plev[nz] = P(z);
00102     if ((++nz) > NZ)
00103         ERRMSG("Too many pressure levels!");
00104 }
00105
00106 /* Set horizontal grid... */
00107 if (dlon <= 0)
00108     dlon = fabs(met->lon[1] - met->lon[0]);
00109 if (dlat <= 0)
00110     dlat = fabs(met->lat[1] - met->lat[0]);
00111
00112 /* Average... */
00113 for (iz = 0; iz < nz; iz++)
00114     for (lon = lon0; lon <= lon1; lon += dlon)
00115         for (lat = lat0; lat <= lat1; lat += dlat) {
00116
00117             /* Interpolate meteo data... */
00118             INTPOL_SPACE_ALL(plev[iz], lon, lat);
00119
00120             /* Averaging... */
00121             if (gsl_finite(t) && gsl_finite(u)
00122                 && gsl_finite(v) && gsl_finite(w)) {
00123                 timem[iz] += met->time;
00124                 lonm[iz] += lon;
00125                 latm[iz] += lat;
00126                 zm[iz] += z;
00127                 tm[iz] += t;
00128                 um[iz] += u;
00129                 vm[iz] += v;
00130                 wm[iz] += w;
00131                 pvm[iz] += pv;
00132                 h2om[iz] += h2o;
00133                 o3m[iz] += o3;
00134                 lwcm[iz] += lwc;
00135                 iwcm[iz] += iwc;
00136                 psm[iz] += ps;
00137                 tsm[iz] += ts;
00138                 zsm[iz] += zs;
00139                 usm[iz] += us;
00140                 vsm[iz] += vs;
00141                 pblm[iz] += pbl;
00142                 pctm[iz] += pct;
00143                 pcbm[iz] += pcb;
00144                 clm[iz] += cl;
00145                 if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00146                     && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00147                     plclm[iz] += plcl;
00148                     plfcm[iz] += plfc;
00149                     pelm[iz] += pel;
00150                     capem[iz] += cape;
00151                     cinm[iz] += cin;
00152                     npc[iz]++;
00153                 }
00154                 if (gsl_finite(pt)) {
00155                     ptm[iz] += pt;
00156                     ztm[iz] += zt;
00157                     ttm[iz] += tt;
00158                     h2otm[iz] += h2ot;
00159                     npt[iz]++;
00160                 }
00161                 rhm[iz] += RH(plev[iz], t, h2o);
00162                 rhicem[iz] += RHICE(plev[iz], t, h2o);
00163                 tdewm[iz] += TDEW(plev[iz], h2o);
00164                 ticem[iz] += TICE(plev[iz], h2o);
00165                 hno3m[iz] += clim_hno3(met->time, lat, plev[iz]);
00166                 tnatm[iz] +=
00167                     nat_temperature(plev[iz], h2o,
00168                                     clim_hno3(met->time, lat, plev[iz]));
00169                 ohm[iz] += clim_oh(met->time, lat, plev[iz]);
00170                 np[iz]++;
00171             }
00172         }
00173 }
00174
00175 /* Create output file... */
00176 LOG(1, "Write meteorological data file: %s", argv[2]);
00177 if (!(out = fopen(argv[2], "w")))
00178     ERRMSG("Cannot create file!");
00179
00180 /* Write header... */
00181 fprintf(out,
00182     "# $1 = time [s]\n"
00183     "# $2 = altitude [km]\n"
00184     "# $3 = longitude [deg]\n"
00185     "# $4 = latitude [deg]\n"
00186     "# $5 = pressure [hPa]\n"
00187     "# $6 = temperature [K]\n"

```

```

00188         "# $7 = zonal wind [m/s]\n"
00189         "# $8 = meridional wind [m/s]\n"
00190         "# $9 = vertical velocity [hPa/s]\n"
00191         "# $10 = H2O volume mixing ratio [ppv]\n");
00192     fprintf(out,
00193         "# $11 = O3 volume mixing ratio [ppv]\n"
00194         "# $12 = geopotential height [km]\n"
00195         "# $13 = potential vorticity [PVU]\n"
00196         "# $14 = surface pressure [hPa]\n"
00197         "# $15 = surface temperature [K]\n"
00198         "# $16 = surface geopotential height [km]\n"
00199         "# $17 = surface zonal wind [m/s]\n"
00200         "# $18 = surface meridional wind [m/s]\n"
00201         "# $19 = tropopause pressure [hPa]\n"
00202         "# $20 = tropopause geopotential height [km]\n");
00203     fprintf(out,
00204         "# $21 = tropopause temperature [K]\n"
00205         "# $22 = tropopause water vapor [ppv]\n"
00206         "# $23 = cloud liquid water content [kg/kg]\n"
00207         "# $24 = cloud ice water content [kg/kg]\n"
00208         "# $25 = total column cloud water [kg/m^2]\n"
00209         "# $26 = cloud top pressure [hPa]\n"
00210         "# $27 = cloud bottom pressure [hPa]\n"
00211         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00212         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00213         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00214     fprintf(out,
00215         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00216         "# $32 = convective inhibition (CIN) [J/kg]\n"
00217         "# $33 = relative humidity over water [%]\n"
00218         "# $34 = relative humidity over ice [%]\n"
00219         "# $35 = dew point temperature [K]\n"
00220         "# $36 = frost point temperature [K]\n"
00221         "# $37 = NAT temperature [K]\n"
00222         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00223         "# $39 = OH concentration [molec/cm^3]\n"
00224         "# $40 = boundary layer pressure [hPa]\n");
00225     fprintf(out,
00226         "# $41 = number of data points\n"
00227         "# $42 = number of tropopause data points\n"
00228         "# $43 = number of CAPE data points\n\n");
00229
00230     /* Write data... */
00231     for (iz = 0; iz < nz; iz++)
00232         fprintf(out,
00233             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n"
00234             " %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00235             " %g %g %g %g %g %g %g %g %g %d %d %d\n",
00236             timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00237             latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00238             vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00239             o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00240             psm[iz] / np[iz], tsm[iz] / np[iz], zsm[iz] / np[iz],
00241             usm[iz] / np[iz], vsm[iz] / np[iz], ptm[iz] / npt[iz],
00242             ztm[iz] / npt[iz], ttm[iz] / npt[iz], h2otm[iz] / npt[iz],
00243             lwcm[iz] / np[iz], iwcm[iz] / np[iz], clm[iz] / np[iz],
00244             pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
00245             plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
00246             cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
00247             tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
00248             hno3m[iz] / np[iz], ohm[iz] / np[iz], pblm[iz] / np[iz],
00249             np[iz], npt[iz], npc[iz]);
00250
00251     /* Close file... */
00252     fclose(out);
00253
00254     /* Free... */
00255     free(met);
00256
00257     return EXIT_SUCCESS;
00258 }

```

5.29 met_sample.c File Reference

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

Functions

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

5.29.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file [met_sample.c](#).

5.29.2 Function Documentation

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

Definition at line 31 of file [met_sample.c](#).

```
00033     {
00034
00035     ctl_t ctl;
00036
00037     atm_t *atm;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     double h2o, h2ot, o3, lwc, iwc, p0, pl, ps, ts, zs, us, vs, pbl, pt,
00044           pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00045           zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00046
00047     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00048
00049     /* Check arguments... */
00050     if (argc < 3)
00051         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00052
00053     /* Allocate... */
00054     ALLOC(atm, atm_t, 1);
00055     ALLOC(met0, met_t, 1);
00056     ALLOC(met1, met_t, 1);
00057
00058     /* Read control parameters... */
00059     read_ctl(argv[1], argc, argv, &ctl);
00060     geopot =
00061         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00062     grid_time =
00063         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00064     grid_z =
00065         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00066     grid_lon =
00067         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00068     grid_lat =
00069         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00070
00071     /* Read atmospheric data... */
00072     if (!read_atm(argv[3], &ctl, atm))
00073         ERRMSG("Cannot open file!");
00074
00075     /* Create output file... */
00076     LOG(1, "Write meteorological data file: %s", argv[2]);
00077     if (!(out = fopen(argv[2], "w")))
00078         ERRMSG("Cannot create file!");
00079
00080     /* Write header... */
00081     fprintf(out,
00082           "# $1 = time [s]\n"
00083           "# $2 = altitude [km]\n"
00084           "# $3 = longitude [deg]\n"
00085           "# $4 = latitude [deg]\n"
00086           "# $5 = pressure [hPa]\n"
00087           "# $6 = temperature [K]\n"
00088           "# $7 = zonal wind [m/s]\n"
00089           "# $8 = meridional wind [m/s]\n"
00090           "# $9 = vertical velocity [hPa/s]\n"
00091           "# $10 = H2O volume mixing ratio [ppv]\n");
00092     fprintf(out,
```



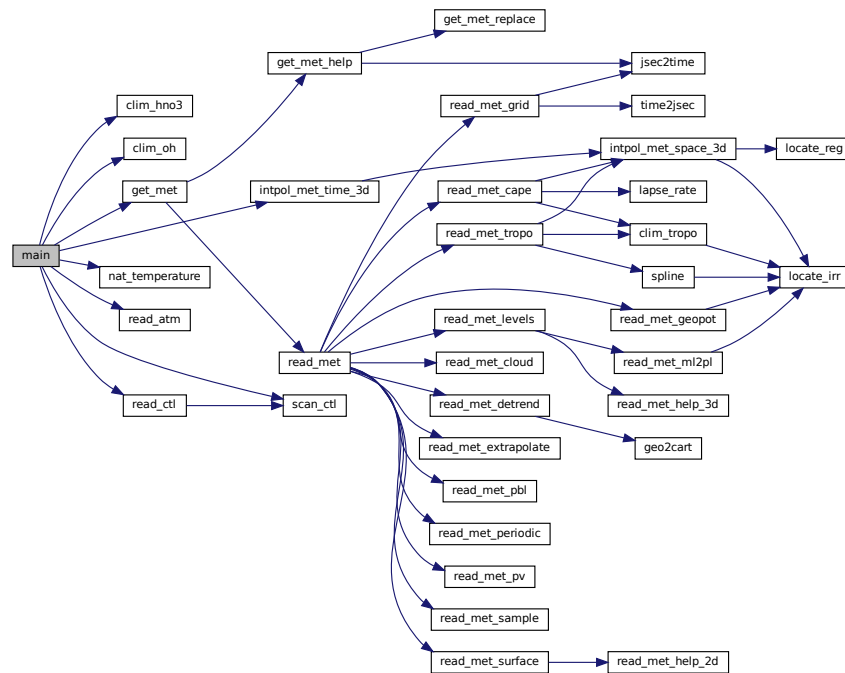
```
00093 "# $1 = O3 volume mixing ratio [ppv]\n"
00094 "# $12 = geopotential height [km]\n"
00095 "# $13 = potential vorticity [PVU]\n"
00096 "# $14 = surface pressure [hPa]\n"
00097 "# $15 = surface temperature [K]\n"
00098 "# $16 = surface geopotential height [km]\n"
00099 "# $17 = surface zonal wind [m/s]\n"
00100 "# $18 = surface meridional wind [m/s]\n"
00101 "# $19 = tropopause pressure [hPa]\n"
00102 "# $20 = tropopause geopotential height [km]\n");
00103 fprintf(out,
00104 "# $21 = tropopause temperature [K]\n"
00105 "# $22 = tropopause water vapor [ppv]\n"
00106 "# $23 = cloud liquid water content [kg/kg]\n"
00107 "# $24 = cloud ice water content [kg/kg]\n"
00108 "# $25 = total column cloud water [kg/m^2]\n"
00109 "# $26 = cloud top pressure [hPa]\n"
00110 "# $27 = cloud bottom pressure [hPa]\n"
00111 "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00112 "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00113 "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00114 fprintf(out,
00115 "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00116 "# $32 = convective inhibition (CIN) [J/kg]\n"
00117 "# $33 = relative humidity over water [%]\n"
00118 "# $34 = relative humidity over ice [%]\n"
00119 "# $35 = dew point temperature [K]\n"
00120 "# $36 = frost point temperature [K]\n"
00121 "# $37 = NAT temperature [K]\n"
00122 "# $38 = HNO3 volume mixing ratio [ppv]\n"
00123 "# $39 = OH concentration [molec/cm^3]\n"
00124 "# $40 = boundary layer pressure [hPa]\n");
00125
00126 /* Loop over air parcels... */
00127 for (ip = 0; ip < atm->np; ip++) {
00128
00129     /* Get meteorological data... */
00130     get_met(&ctl, atm->time[ip], &met0, &met1);
00131
00132     /* Set reference pressure for interpolation... */
00133     double pref = atm->p[ip];
00134     if (geopot) {
00135         zref = Z(pref);
00136         p0 = met0->p[0];
00137         pl = met0->p[met0->np - 1];
00138         for (it = 0; it < 24; it++) {
00139             pref = 0.5 * (p0 + pl);
00140             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
00141                               atm->lon[ip], atm->lat[ip], &zm, ci, cw, l);
00142             if (zref > zm || !gsl_finite(zm))
00143                 p0 = pref;
00144             else
00145                 pl = pref;
00146         }
00147         pref = 0.5 * (p0 + pl);
00148     }
00149
00150     /* Interpolate meteo data... */
00151     INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
00152
00153     /* Make blank lines... */
00154     if (ip == 0 || (grid_time && atm->time[ip] != time_old)
00155         || (grid_z && atm->p[ip] != p_old)
00156         || (grid_lon && atm->lon[ip] != lon_old)
00157         || (grid_lat && atm->lat[ip] != lat_old))
00158         fprintf(out, "\n");
00159     time_old = atm->time[ip];
00160     p_old = atm->p[ip];
00161     lon_old = atm->lon[ip];
00162     lat_old = atm->lat[ip];
00163
00164     /* Write data... */
00165     fprintf(out,
00166            "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00167            atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00168            atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
00169            pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
00170            cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
00171            TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
00172            nat_temperature(atm->p[ip], h2o,
00173                           clim_hno3(atm->time[ip], atm->lat[ip],
00174                                     atm->p[ip])), clim_hno3(atm->time[ip],
00175                                                                atm->lat[ip],
00176                                                                atm->p[ip]),
00177            clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
00178 }
00179
```

```

00180
00181  /* Close file... */
00182  fclose(out);
00183
00184  /* Free... */
00185  free(atm);
00186  free(met0);
00187  free(met1);
00188
00189  return EXIT_SUCCESS;
00190 }

```

Here is the call graph for this function:



5.30 met_sample.c

```

00001  /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  /* -----
00028   Main...
00029   ----- */
00030
00031  int main(
00032      int argc,
00033      char *argv[]) {
00034

```

```

00035     ctl_t ctl;
00036
00037     atm_t *atm;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     double h2o, h2ot, o3, lwc, iwc, p0, p1, ps, ts, zs, us, vs, pbl, pt,
00044           pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00045           zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00046
00047     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00048
00049     /* Check arguments... */
00050     if (argc < 3)
00051         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00052
00053     /* Allocate... */
00054     ALLOC(atm, atm_t, 1);
00055     ALLOC(met0, met_t, 1);
00056     ALLOC(met1, met_t, 1);
00057
00058     /* Read control parameters... */
00059     read_ctl(argv[1], argc, argv, &ctl);
00060     geopot =
00061         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00062     grid_time =
00063         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00064     grid_z =
00065         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00066     grid_lon =
00067         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00068     grid_lat =
00069         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00070
00071     /* Read atmospheric data... */
00072     if (!read_atm(argv[3], &ctl, atm))
00073         ERRMSG("Cannot open file!");
00074
00075     /* Create output file... */
00076     LOG(1, "Write meteorological data file: %s", argv[2]);
00077     if (!(out = fopen(argv[2], "w")))
00078         ERRMSG("Cannot create file!");
00079
00080     /* Write header... */
00081     fprintf(out,
00082            "# $1 = time [s]\n"
00083            "# $2 = altitude [km]\n"
00084            "# $3 = longitude [deg]\n"
00085            "# $4 = latitude [deg]\n"
00086            "# $5 = pressure [hPa]\n"
00087            "# $6 = temperature [K]\n"
00088            "# $7 = zonal wind [m/s]\n"
00089            "# $8 = meridional wind [m/s]\n"
00090            "# $9 = vertical velocity [hPa/s]\n"
00091            "# $10 = H2O volume mixing ratio [ppv]\n");
00092     fprintf(out,
00093            "# $11 = O3 volume mixing ratio [ppv]\n"
00094            "# $12 = geopotential height [km]\n"
00095            "# $13 = potential vorticity [PVU]\n"
00096            "# $14 = surface pressure [hPa]\n"
00097            "# $15 = surface temperature [K]\n"
00098            "# $16 = surface geopotential height [km]\n"
00099            "# $17 = surface zonal wind [m/s]\n"
00100            "# $18 = surface meridional wind [m/s]\n"
00101            "# $19 = tropopause pressure [hPa]\n"
00102            "# $20 = tropopause geopotential height [km]\n");
00103     fprintf(out,
00104            "# $21 = tropopause temperature [K]\n"
00105            "# $22 = tropopause water vapor [ppv]\n"
00106            "# $23 = cloud liquid water content [kg/kg]\n"
00107            "# $24 = cloud ice water content [kg/kg]\n"
00108            "# $25 = total column cloud water [kg/m^2]\n"
00109            "# $26 = cloud top pressure [hPa]\n"
00110            "# $27 = cloud bottom pressure [hPa]\n"
00111            "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00112            "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00113            "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00114     fprintf(out,
00115            "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00116            "# $32 = convective inhibition (CIN) [J/kg]\n"
00117            "# $33 = relative humidity over water [%]\n"
00118            "# $34 = relative humidity over ice [%]\n"
00119            "# $35 = dew point temperature [K]\n"
00120            "# $36 = frost point temperature [K]\n"
00121            "# $37 = NAT temperature [K]\n");

```


Functions

- void [fft_help](#) (double *fcReal, double *fcImag, int n)
- int [main](#) (int argc, char *argv[])

5.31.1 Detailed Description

Spectral analysis of meteorological data.

Definition in file [met_spec.c](#).

5.31.2 Macro Definition Documentation

5.31.2.1 PMAX `#define PMAX EX`

Maximum number of data points for spectral analysis.

Definition at line 32 of file [met_spec.c](#).

5.31.3 Function Documentation

5.31.3.1 [fft_help\(\)](#) void [fft_help](#) (

```
double * fcReal,
double * fcImag,
int n )
```

Definition at line 143 of file [met_spec.c](#).

```
00146     {
00147
00148     gsl_fft_complex_wavetable *wavetable;
00149     gsl_fft_complex_workspace *workspace;
00150
00151     double data[2 * PMAX];
00152
00153     int i;
00154
00155     /* Check size... */
00156     if (n > PMAX)
00157         ERRMSG("Too many data points!");
00158
00159     /* Allocate... */
00160     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00161     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00162
00163     /* Set data (real, complex)... */
00164     for (i = 0; i < n; i++) {
00165         data[2 * i] = fcReal[i];
00166         data[2 * i + 1] = fcImag[i];
00167     }
00168
00169     /* Calculate FFT... */
00170     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00171
00172     /* Copy data... */
00173     for (i = 0; i < n; i++) {
00174         fcReal[i] = data[2 * i];
00175         fcImag[i] = data[2 * i + 1];
00176     }
00177
00178     /* Free... */
00179     gsl_fft_complex_wavetable_free(wavetable);
00180     gsl_fft_complex_workspace_free(workspace);
00181 }
```

```

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

```

Definition at line 47 of file `met_spec.c`.

```

00049     {
00050
00051         ctl_t ctl;
00052
00053         met_t *met;
00054
00055         FILE *out;
00056
00057         static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00058             wavemax;
00059
00060         /* Allocate... */
00061         ALLOC(met, met_t, 1);
00062
00063         /* Check arguments... */
00064         if (argc < 4)
00065             ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00066
00067         /* Read control parameters... */
00068         read_ctl(argv[1], argc, argv, &ctl);
00069         wavemax =
00070             (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00071
00072         /* Read meteorological data... */
00073         if (!read_met(&ctl, argv[3], met))
00074             ERRMSG("Cannot read meteo data!");
00075
00076         /* Create output file... */
00077         printf("Write spectral data file: %s\n", argv[2]);
00078         if (!(out = fopen(argv[2], "w")))
00079             ERRMSG("Cannot create file!");
00080
00081         /* Write header... */
00082         fprintf(out,
00083             "# $1 = time [s]\n"
00084             "# $2 = altitude [km]\n"
00085             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00086         for (int ix = 0; ix <= wavemax; ix++) {
00087             fprintf(out, "# $d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
00088             fprintf(out, "# $d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
00089             fprintf(out, "# $d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
00090         }
00091
00092         /* Loop over pressure levels... */
00093         for (int ip = 0; ip < met->np; ip++) {
00094
00095             /* Write output... */
00096             fprintf(out, "\n");
00097
00098             /* Loop over latitudes... */
00099             for (int iy = 0; iy < met->ny; iy++) {
00100
00101                 /* Copy data... */
00102                 for (int ix = 0; ix < met->nx; ix++) {
00103                     cutReal[ix] = met->t[ix][iy][ip];
00104                     cutImag[ix] = 0.0;
00105                 }
00106
00107                 /* FFT... */
00108                 fft_help(cutReal, cutImag, met->nx);
00109
00110                 /*
00111                  * Get wavelength, amplitude, and phase:
00112                  * A(x) = A[0] + A[1] * cos(2 pi x / lx[1] + phi[1]) + A[2] * cos...
00113                  */
00114                 for (int ix = 0; ix < met->nx; ix++) {
00115                     lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00116                         / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00117                     A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00118                         * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00119                     phi[ix]
00120                         = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00121                 }
00122
00123                 /* Write data... */
00124                 fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
00125                     met->lat[iy]);
00126                 for (int ix = 0; ix <= wavemax; ix++)
00127                     fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00128                 fprintf(out, "\n");

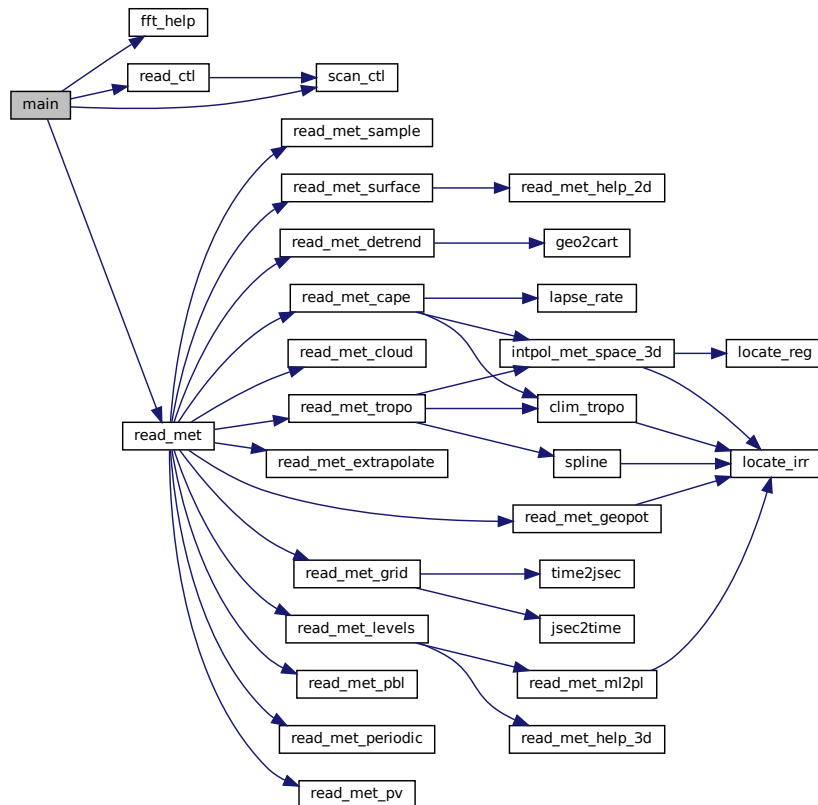
```

```

00129     }
00130 }
00131
00132 /* Close file... */
00133 fclose(out);
00134
00135 /* Free... */
00136 free(met);
00137
00138 return EXIT_SUCCESS;
00139 }

```

Here is the call graph for this function:



5.32 met_spec.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) 2013–2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026

```

```

00027 /* -----
00028     Dimensions...
00029     ----- */
00030
00032 #define PMAX EX
00033
00034 /* -----
00035     Functions...
00036     ----- */
00037
00038 void fft_help(
00039     double *fcReal,
00040     double *fcImag,
00041     int n);
00042
00043 /* -----
00044     Main...
00045     ----- */
00046
00047 int main(
00048     int argc,
00049     char *argv[]) {
00050
00051     ctl_t ctl;
00052
00053     met_t *met;
00054
00055     FILE *out;
00056
00057     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00058         wavemax;
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     wavemax =
00070         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00071
00072     /* Read meteorological data... */
00073     if (!read_met(&ctl, argv[3], met))
00074         ERRMSG("Cannot read meteo data!");
00075
00076     /* Create output file... */
00077     printf("Write spectral data file: %s\n", argv[2]);
00078     if (!(out = fopen(argv[2], "w")))
00079         ERRMSG("Cannot create file!");
00080
00081     /* Write header... */
00082     fprintf(out,
00083         "# $1 = time [s]\n"
00084         "# $2 = altitude [km]\n"
00085         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00086     for (int ix = 0; ix <= wavemax; ix++) {
00087         fprintf(out, "# $d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
00088         fprintf(out, "# $d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
00089         fprintf(out, "# $d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
00090     }
00091
00092     /* Loop over pressure levels... */
00093     for (int ip = 0; ip < met->np; ip++) {
00094
00095         /* Write output... */
00096         fprintf(out, "\n");
00097
00098         /* Loop over latitudes... */
00099         for (int iy = 0; iy < met->ny; iy++) {
00100
00101             /* Copy data... */
00102             for (int ix = 0; ix < met->nx; ix++) {
00103                 cutReal[ix] = met->t[ix][iy][ip];
00104                 cutImag[ix] = 0.0;
00105             }
00106
00107             /* FFT... */
00108             fft_help(cutReal, cutImag, met->nx);
00109
00110             /*
00111              * Get wavelength, amplitude, and phase:
00112              * A(x) = A[0] + A[1] * cos(2 pi x / lx[1] + phi[1]) + A[2] * cos...
00113              */
00114             for (int ix = 0; ix < met->nx; ix++) {

```



```

00115         lx[ix] = DEG2DX (met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00116         / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00117         A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00118         * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00119         phi[ix]
00120         = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00121     }
00122
00123     /* Write data... */
00124     fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
00125            met->lat[iy]);
00126     for (int ix = 0; ix <= wavemax; ix++)
00127         fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00128     fprintf(out, "\n");
00129 }
00130 }
00131
00132 /* Close file... */
00133 fclose(out);
00134
00135 /* Free... */
00136 free(met);
00137
00138 return EXIT_SUCCESS;
00139 }
00140
00141 /*****
00142
00143 void fft_help(
00144     double *fcReal,
00145     double *fcImag,
00146     int n) {
00147
00148     gsl_fft_complex_wavetable *wavetable;
00149     gsl_fft_complex_workspace *workspace;
00150
00151     double data[2 * PMAX];
00152
00153     int i;
00154
00155     /* Check size... */
00156     if (n > PMAX)
00157         ERRMSG("Too many data points!");
00158
00159     /* Allocate... */
00160     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00161     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00162
00163     /* Set data (real, complex)... */
00164     for (i = 0; i < n; i++) {
00165         data[2 * i] = fcReal[i];
00166         data[2 * i + 1] = fcImag[i];
00167     }
00168
00169     /* Calculate FFT... */
00170     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00171
00172     /* Copy data... */
00173     for (i = 0; i < n; i++) {
00174         fcReal[i] = data[2 * i];
00175         fcImag[i] = data[2 * i + 1];
00176     }
00177
00178     /* Free... */
00179     gsl_fft_complex_wavetable_free(wavetable);
00180     gsl_fft_complex_workspace_free(workspace);
00181 }

```

5.33 met_subgrid.c File Reference

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

Functions

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

5.33.1 Detailed Description

Calculate standard deviations of horizontal wind and vertical velocity.

Definition in file [met_subgrid.c](#).

5.33.2 Function Documentation

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

Definition at line 31 of file [met_subgrid.c](#).

```
00033 {
00034
00035     ctl_t ctl;
00036
00037     met_t *met0, *met1;
00038
00039     FILE *out;
00040
00041     static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];
00042
00043     static float u[16], v[16], w[16];
00044
00045     static int i, ix, iy, iz, n[EP][EY];
00046
00047     /* Allocate... */
00048     ALLOC(met0, met_t, 1);
00049     ALLOC(met1, met_t, 1);
00050
00051     /* Check arguments... */
00052     if (argc < 4 && argc % 2 != 0)
00053         ERRMSG
00054             ("Give parameters: <ctl> <zm.tab> <met0> <met1> [ <met0> <met1> ... ]");
00055
00056     /* Read control parameters... */
00057     read_ctl(argv[1], argc, argv, &ctl);
00058
00059     /* Loop over data files... */
00060     for (i = 3; i < argc - 1; i += 2) {
00061
00062         /* Read meteorological data... */
00063         if (!read_met(&ctl, argv[i], met0))
00064             ERRMSG("Cannot open file!");
00065         if (!read_met(&ctl, argv[i + 1], met1))
00066             ERRMSG("Cannot open file!");
00067
00068         /* Loop over grid boxes... */
00069         for (ix = 0; ix < met0->nx - 1; ix++)
00070             for (iy = 0; iy < met0->ny - 1; iy++)
00071                 for (iz = 0; iz < met0->np - 1; iz++) {
00072
00073                     /* Collect local wind data... */
00074                     u[0] = met0->u[ix][iy][iz];
00075                     u[1] = met0->u[ix + 1][iy][iz];
00076                     u[2] = met0->u[ix][iy + 1][iz];
00077                     u[3] = met0->u[ix + 1][iy + 1][iz];
00078                     u[4] = met0->u[ix][iy][iz + 1];
00079                     u[5] = met0->u[ix + 1][iy][iz + 1];
00080                     u[6] = met0->u[ix][iy + 1][iz + 1];
00081                     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00082
00083                     v[0] = met0->v[ix][iy][iz];
00084                     v[1] = met0->v[ix + 1][iy][iz];
00085                     v[2] = met0->v[ix][iy + 1][iz];
00086                     v[3] = met0->v[ix + 1][iy + 1][iz];
00087                     v[4] = met0->v[ix][iy][iz + 1];
00088                     v[5] = met0->v[ix + 1][iy][iz + 1];
00089                     v[6] = met0->v[ix][iy + 1][iz + 1];
00090                     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00091
00092                     w[0] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
```

```

00093     w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
00094     w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
00095     w[3] =
00096         (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
00097     w[4] =
00098         (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
00099     w[5] =
00100         (float) (1e3 *
00101             DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
00102     w[6] =
00103         (float) (1e3 *
00104             DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
00105     w[7] =
00106         (float) (1e3 *
00107             DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[iz + 1]));
00108
00109     /* Collect local wind data... */
00110     u[8] = met1->u[ix][iy][iz];
00111     u[9] = met1->u[ix + 1][iy][iz];
00112     u[10] = met1->u[ix][iy + 1][iz];
00113     u[11] = met1->u[ix + 1][iy + 1][iz];
00114     u[12] = met1->u[ix][iy][iz + 1];
00115     u[13] = met1->u[ix + 1][iy][iz + 1];
00116     u[14] = met1->u[ix][iy + 1][iz + 1];
00117     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00118
00119     v[8] = met1->v[ix][iy][iz];
00120     v[9] = met1->v[ix + 1][iy][iz];
00121     v[10] = met1->v[ix][iy + 1][iz];
00122     v[11] = met1->v[ix + 1][iy + 1][iz];
00123     v[12] = met1->v[ix][iy][iz + 1];
00124     v[13] = met1->v[ix + 1][iy][iz + 1];
00125     v[14] = met1->v[ix][iy + 1][iz + 1];
00126     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00127
00128     w[8] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
00129     w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
00130     w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
00131     w[11] =
00132         (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
00133     w[12] =
00134         (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
00135     w[13] =
00136         (float) (1e3 *
00137             DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
00138     w[14] =
00139         (float) (1e3 *
00140             DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
00141     w[15] =
00142         (float) (1e3 *
00143             DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));
00144
00145     /* Get standard deviations of local wind data... */
00146     usig[ix][iy] += stddev(u, 16);
00147     vsig[ix][iy] += stddev(v, 16);
00148     wsig[ix][iy] += stddev(w, 16);
00149     n[ix][iy]++;
00150
00151     /* Check surface pressure... */
00152     if (met0->p[iz] > met0->ps[ix][iy]
00153         || met1->p[iz] > met1->ps[ix][iy]) {
00154         usig[ix][iy] = GSL_NAN;
00155         vsig[ix][iy] = GSL_NAN;
00156         wsig[ix][iy] = GSL_NAN;
00157         n[ix][iy] = 0;
00158     }
00159 }
00160 }
00161
00162 /* Create output file... */
00163 LOG(1, "Write subgrid data file: %s", argv[2]);
00164 if (!out = fopen(argv[2], "w"))
00165     ERRMSG("Cannot create file!");
00166
00167 /* Write header... */
00168 fprintf(out,
00169     "# $1 = time [s]\n"
00170     "# $2 = altitude [km]\n"
00171     "# $3 = longitude [deg]\n"
00172     "# $4 = latitude [deg]\n"
00173     "# $5 = zonal wind standard deviation [m/s]\n"
00174     "# $6 = meridional wind standard deviation [m/s]\n"
00175     "# $7 = vertical velocity standard deviation [m/s]\n"
00176     "# $8 = number of data points\n");
00177
00178 /* Write output... */
00179 for (iy = 0; iy < met0->ny - 1; iy++) {

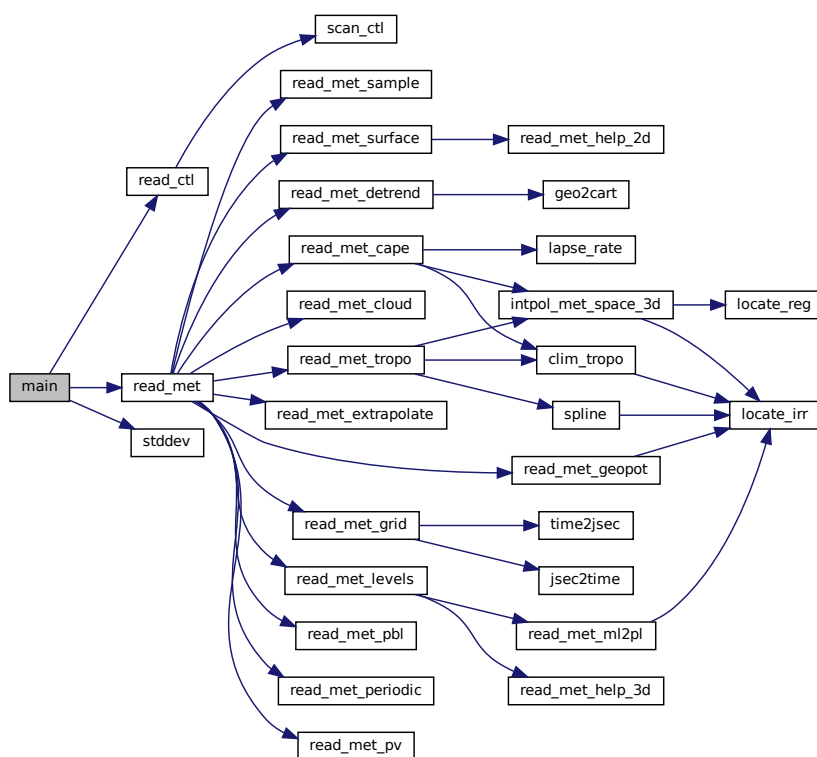
```

```

00180     fprintf(out, "\n");
00181     for (iz = 0; iz < met0->np - 1; iz++)
00182         fprintf(out, "%.2f %g %g %g %g %g %g %d\n",
00183             0.5 * (met0->time + met1->time),
00184             0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
00185             0.0, 0.5 * (met0->lat[iz] + met1->lat[iz + 1]),
00186             usig[iz][iz] / n[iz][iz], vsig[iz][iz] / n[iz][iz],
00187             wsig[iz][iz] / n[iz][iz], n[iz][iz]);
00188     }
00189     /* Close file... */
00190     fclose(out);
00191
00192     /* Free... */
00193     free(met0);
00194     free(met1);
00195
00196     return EXIT_SUCCESS;
00197 }
00198

```

Here is the call graph for this function:



5.34 met_subgrid.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) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028 Main...
00029 ----- */
00030
00031 int main(
00032     int argc,
00033     char *argv[]) {
00034
00035     ctl_t ctl;
00036
00037     met_t *met0, *met1;
00038
00039     FILE *out;
00040
00041     static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];
00042
00043     static float u[16], v[16], w[16];
00044
00045     static int i, ix, iy, iz, n[EP][EY];
00046
00047     /* Allocate... */
00048     ALLOC(met0, met_t, 1);
00049     ALLOC(met1, met_t, 1);
00050
00051     /* Check arguments... */
00052     if (argc < 4 && argc % 2 != 0)
00053         ERRMSG
00054             ("Give parameters: <ctl> <zm.tab> <met0> <met1> [ <met0> <met1> ... ]");
00055
00056     /* Read control parameters... */
00057     read_ctl(argv[1], argc, argv, &ctl);
00058
00059     /* Loop over data files... */
00060     for (i = 3; i < argc - 1; i += 2) {
00061
00062         /* Read meteorological data... */
00063         if (!read_met(&ctl, argv[i], met0))
00064             ERRMSG("Cannot open file!");
00065         if (!read_met(&ctl, argv[i + 1], met1))
00066             ERRMSG("Cannot open file!");
00067
00068         /* Loop over grid boxes... */
00069         for (ix = 0; ix < met0->nx - 1; ix++)
00070             for (iy = 0; iy < met0->ny - 1; iy++)
00071                 for (iz = 0; iz < met0->np - 1; iz++) {
00072
00073                     /* Collect local wind data... */
00074                     u[0] = met0->u[ix][iy][iz];
00075                     u[1] = met0->u[ix + 1][iy][iz];
00076                     u[2] = met0->u[ix][iy + 1][iz];
00077                     u[3] = met0->u[ix + 1][iy + 1][iz];
00078                     u[4] = met0->u[ix][iy][iz + 1];
00079                     u[5] = met0->u[ix + 1][iy][iz + 1];
00080                     u[6] = met0->u[ix][iy + 1][iz + 1];
00081                     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00082
00083                     v[0] = met0->v[ix][iy][iz];
00084                     v[1] = met0->v[ix + 1][iy][iz];
00085                     v[2] = met0->v[ix][iy + 1][iz];
00086                     v[3] = met0->v[ix + 1][iy + 1][iz];
00087                     v[4] = met0->v[ix][iy][iz + 1];
00088                     v[5] = met0->v[ix + 1][iy][iz + 1];
00089                     v[6] = met0->v[ix][iy + 1][iz + 1];
00090                     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00091
00092                     w[0] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
00093                     w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
00094                     w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
00095                     w[3] =
00096                         (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
00097                     w[4] =
00098                         (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
00099                     w[5] =
00100                         (float) (1e3 *
00101                             DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
00102                     w[6] =
00103                         (float) (1e3 *
00104                             DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
00105                     w[7] =
00106                         (float) (1e3 *
00107                             DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[iz + 1]));
00108

```

```

00109      /* Collect local wind data... */
00110      u[8] = met1->u[ix][iy][iz];
00111      u[9] = met1->u[ix + 1][iy][iz];
00112      u[10] = met1->u[ix][iy + 1][iz];
00113      u[11] = met1->u[ix + 1][iy + 1][iz];
00114      u[12] = met1->u[ix][iy][iz + 1];
00115      u[13] = met1->u[ix + 1][iy][iz + 1];
00116      u[14] = met1->u[ix][iy + 1][iz + 1];
00117      u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00118
00119      v[8] = met1->v[ix][iy][iz];
00120      v[9] = met1->v[ix + 1][iy][iz];
00121      v[10] = met1->v[ix][iy + 1][iz];
00122      v[11] = met1->v[ix + 1][iy + 1][iz];
00123      v[12] = met1->v[ix][iy][iz + 1];
00124      v[13] = met1->v[ix + 1][iy][iz + 1];
00125      v[14] = met1->v[ix][iy + 1][iz + 1];
00126      v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00127
00128      w[8] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
00129      w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
00130      w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
00131      w[11] =
00132          (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
00133      w[12] =
00134          (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
00135      w[13] =
00136          (float) (1e3 *
00137              DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
00138      w[14] =
00139          (float) (1e3 *
00140              DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
00141      w[15] =
00142          (float) (1e3 *
00143              DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));
00144
00145      /* Get standard deviations of local wind data... */
00146      usig[iz][iy] += stddev(u, 16);
00147      vsig[iz][iy] += stddev(v, 16);
00148      wsig[iz][iy] += stddev(w, 16);
00149      n[iz][iy]++;
00150
00151      /* Check surface pressure... */
00152      if (met0->p[iz] > met0->ps[ix][iy]
00153          || met1->p[iz] > met1->ps[ix][iy]) {
00154          usig[iz][iy] = GSL_NAN;
00155          vsig[iz][iy] = GSL_NAN;
00156          wsig[iz][iy] = GSL_NAN;
00157          n[iz][iy] = 0;
00158      }
00159  }
00160 }
00161
00162 /* Create output file... */
00163 LOG(1, "Write subgrid data file: %s", argv[2]);
00164 if (!(out = fopen(argv[2], "w")))
00165     ERRMSG("Cannot create file!");
00166
00167 /* Write header... */
00168 fprintf(out,
00169     "# $1 = time [s]\n"
00170     "# $2 = altitude [km]\n"
00171     "# $3 = longitude [deg]\n"
00172     "# $4 = latitude [deg]\n"
00173     "# $5 = zonal wind standard deviation [m/s]\n"
00174     "# $6 = meridional standard deviation [m/s]\n"
00175     "# $7 = vertical velocity standard deviation [m/s]\n"
00176     "# $8 = number of data points\n");
00177
00178 /* Write output... */
00179 for (iy = 0; iy < met0->ny - 1; iy++) {
00180     fprintf(out, "\n");
00181     for (iz = 0; iz < met0->np - 1; iz++)
00182         fprintf(out, "%.2f %g %g %g %g %g %d\n",
00183             0.5 * (met0->time + met1->time),
00184             0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
00185             0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
00186             usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
00187             wsig[iz][iy] / n[iz][iy], n[iz][iy]);
00188 }
00189
00190 /* Close file... */
00191 fclose(out);
00192
00193 /* Free... */
00194 free(met0);
00195 free(met1);

```

```
00196
00197     return EXIT_SUCCESS;
00198 }
```

5.35 met_zm.c File Reference

```
#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[])`

5.35.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file [met_zm.c](#).

5.35.2 Macro Definition Documentation

5.35.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line [32](#) of file [met_zm.c](#).

5.35.2.2 NY `#define NY 721`

Maximum number of latitudes.

Definition at line [35](#) of file [met_zm.c](#).

5.35.3 Function Documentation

```

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

```

Definition at line 41 of file [met_zm.c](#).

```

00043     {
00044
00045         ctl_t ctl;
00046
00047         met_t *met;
00048
00049         FILE *out;
00050
00051         static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00052             usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
00053             pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
00054             capem[NZ][NY], cinm[NZ][NY], ttcm[NZ][NY], ztm[NZ][NY], tm[NZ][NY],
00055             um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY],
00056             pvm[NZ][NY], o3m[NZ][NY], lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY],
00057             rhm[NZ][NY], rhicm[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
00058             hno3m[NZ][NY], ohm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
00059             ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
00060             cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
00061             lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];
00062
00063         static int i, ix, iy, iz, np[NZ][NY], npc[NZ][NY], npt[NZ][NY], ny, nz,
00064             ci[3];
00065
00066         /* Allocate... */
00067         ALLOC(met, met_t, 1);
00068
00069         /* Check arguments... */
00070         if (argc < 4)
00071             ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00072
00073         /* Read control parameters... */
00074         read_ctl(argv[1], argc, argv, &ctl);
00075         z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00076         z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00077         dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00078         lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
00079         lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", NULL);
00080         lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00081         lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00082         dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00083
00084         /* Loop over files... */
00085         for (i = 3; i < argc; i++) {
00086
00087             /* Read meteorological data... */
00088             if (!read_met(&ctl, argv[i], met))
00089                 continue;
00090
00091             /* Set vertical grid... */
00092             if (z0 < 0)
00093                 z0 = Z(met->p[0]);
00094             if (z1 < 0)
00095                 z1 = Z(met->p[met->np - 1]);
00096             nz = 0;
00097             if (dz < 0) {
00098                 for (iz = 0; iz < met->np; iz++)
00099                     if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00100                         plev[nz] = met->p[iz];
00101                         if ((++nz) > NZ)
00102                             ERRMSG("Too many pressure levels!");
00103                     }
00104             } else
00105                 for (z = z0; z <= z1; z += dz) {
00106                     plev[nz] = P(z);
00107                     if ((++nz) > NZ)
00108                         ERRMSG("Too many pressure levels!");
00109                 }
00110
00111             /* Set horizontal grid... */
00112             if (dlat <= 0)
00113                 dlat = fabs(met->lat[1] - met->lat[0]);
00114             ny = 0;
00115             if (lat0 < -90 && lat1 > 90) {
00116                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00117                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00118             }
00119             for (lat = lat0; lat <= lat1; lat += dlat) {
00120                 lats[ny] = lat;
00121                 if ((++ny) > NY)
00122                     ERRMSG("Too many latitudes!");

```



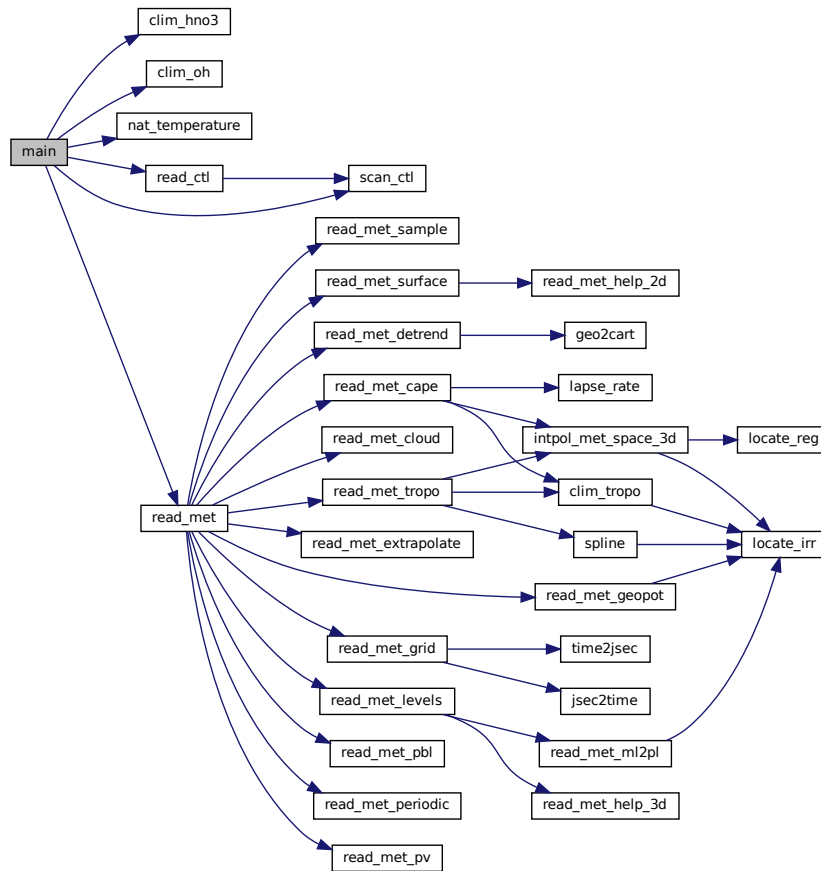
```

00123     }
00124
00125     /* Average... */
00126     for (ix = 0; ix < met->nx; ix++)
00127         if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
00128             for (iy = 0; iy < ny; iy++)
00129                 for (iz = 0; iz < nz; iz++) {
00130
00131                     /* Interpolate meteo data... */
00132                     INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);
00133
00134                     /* Averaging... */
00135                     timem[iz][iy] += met->time;
00136                     lonm[iz][iy] += met->lon[ix];
00137                     zm[iz][iy] += z;
00138                     tm[iz][iy] += t;
00139                     um[iz][iy] += u;
00140                     vm[iz][iy] += v;
00141                     wm[iz][iy] += w;
00142                     pvm[iz][iy] += pv;
00143                     h2om[iz][iy] += h2o;
00144                     o3m[iz][iy] += o3;
00145                     lwcm[iz][iy] += lwc;
00146                     iwcm[iz][iy] += iwc;
00147                     psm[iz][iy] += ps;
00148                     tsm[iz][iy] += ts;
00149                     zsm[iz][iy] += zs;
00150                     usm[iz][iy] += us;
00151                     vsm[iz][iy] += vs;
00152                     pblm[iz][iy] += pbl;
00153                     pctm[iz][iy] += pct;
00154                     pcbm[iz][iy] += pcb;
00155                     clm[iz][iy] += cl;
00156                     if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00157                         && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00158                         plclm[iz][iy] += plcl;
00159                         plfcm[iz][iy] += plfc;
00160                         pelm[iz][iy] += pel;
00161                         capem[iz][iy] += cape;
00162                         cinm[iz][iy] += cin;
00163                         npc[iz][iy]++;
00164                     }
00165                     if (gsl_finite(pt)) {
00166                         ptm[iz][iy] += pt;
00167                         ztm[iz][iy] += zt;
00168                         ttm[iz][iy] += tt;
00169                         h2otm[iz][iy] += h2ot;
00170                         npt[iz][iy]++;
00171                     }
00172                     rhm[iz][iy] += RH(plev[iz], t, h2o);
00173                     rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
00174                     tdewm[iz][iy] += TDEW(plev[iz], h2o);
00175                     ticem[iz][iy] += TICE(plev[iz], h2o);
00176                     hno3m[iz][iy] += clim_hno3(met->time, lats[iy], plev[iz]);
00177                     tnatm[iz][iy] +=
00178                         nat_temperature(plev[iz], h2o,
00179                                         clim_hno3(met->time, lats[iy], plev[iz]));
00180                     ohm[iz][iy] += clim_oh(met->time, lats[iy], plev[iz]);
00181                     np[iz][iy]++;
00182                 }
00183     }
00184
00185     /* Create output file... */
00186     LOG(1, "Write meteorological data file: %s", argv[2]);
00187     if (!out = fopen(argv[2], "w"))
00188         ERRMSG("Cannot create file!");
00189
00190     /* Write header... */
00191     fprintf(out,
00192         "# $1 = time [s]\n"
00193         "# $2 = altitude [km]\n"
00194         "# $3 = longitude [deg]\n"
00195         "# $4 = latitude [deg]\n"
00196         "# $5 = pressure [hPa]\n"
00197         "# $6 = temperature [K]\n"
00198         "# $7 = zonal wind [m/s]\n"
00199         "# $8 = meridional wind [m/s]\n"
00200         "# $9 = vertical velocity [hPa/s]\n"
00201         "# $10 = H2O volume mixing ratio [ppv]\n");
00202     fprintf(out,
00203         "# $11 = O3 volume mixing ratio [ppv]\n"
00204         "# $12 = geopotential height [km]\n"
00205         "# $13 = potential vorticity [PVU]\n"
00206         "# $14 = surface pressure [hPa]\n"
00207         "# $15 = surface temperature [K]\n"
00208         "# $16 = surface geopotential height [km]\n"
00209         "# $17 = surface zonal wind [m/s]\n");

```

Generated by Doxygen

Here is the call graph for this function:



5.36 met_zm.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NZ 1000
00033
00035 #define NY 721
00036
00037 /* -----
00038  Main...

```

```

00039 ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00052         usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
00053         pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
00054         capem[NZ][NY], cinm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY],
00055         um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY],
00056         pvm[NZ][NY], o3m[NZ][NY], lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY],
00057         rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
00058         hno3m[NZ][NY], ohm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
00059         ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
00060         cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
00061         lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];
00062
00063     static int i, ix, iy, iz, np[NZ][NY], npc[NZ][NY], npt[NZ][NY], ny, nz,
00064         ci[3];
00065
00066     /* Allocate... */
00067     ALLOC(met, met_t, 1);
00068
00069     /* Check arguments... */
00070     if (argc < 4)
00071         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00072
00073     /* Read control parameters... */
00074     read_ctl(argv[1], argc, argv, &ctl);
00075     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00076     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00077     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00078     lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
00079     lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", NULL);
00080     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00081     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00082     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00083
00084     /* Loop over files... */
00085     for (i = 3; i < argc; i++) {
00086
00087         /* Read meteorological data... */
00088         if (!read_met(&ctl, argv[i], met))
00089             continue;
00090
00091         /* Set vertical grid... */
00092         if (z0 < 0)
00093             z0 = Z(met->p[0]);
00094         if (z1 < 0)
00095             z1 = Z(met->p[met->np - 1]);
00096         nz = 0;
00097         if (dz < 0) {
00098             for (iz = 0; iz < met->np; iz++)
00099                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00100                     plev[nz] = met->p[iz];
00101                     if ((++nz) > NZ)
00102                         ERRMSG("Too many pressure levels!");
00103                 }
00104             } else
00105                 for (z = z0; z <= z1; z += dz) {
00106                     plev[nz] = P(z);
00107                     if ((++nz) > NZ)
00108                         ERRMSG("Too many pressure levels!");
00109                 }
00110
00111         /* Set horizontal grid... */
00112         if (dlat <= 0)
00113             dlat = fabs(met->lat[1] - met->lat[0]);
00114         ny = 0;
00115         if (lat0 < -90 && lat1 > 90) {
00116             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00117             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00118         }
00119         for (lat = lat0; lat <= lat1; lat += dlat) {
00120             lats[ny] = lat;
00121             if ((++ny) > NY)
00122                 ERRMSG("Too many latitudes!");
00123         }
00124
00125         /* Average... */

```

```

00126     for (ix = 0; ix < met->nx; ix++)
00127     if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
00128     for (iy = 0; iy < ny; iy++)
00129     for (iz = 0; iz < nz; iz++) {
00130
00131         /* Interpolate meteo data... */
00132         INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);
00133
00134         /* Averaging... */
00135         timem[iz][iy] += met->time;
00136         lonm[iz][iy] += met->lon[ix];
00137         zm[iz][iy] += z;
00138         tm[iz][iy] += t;
00139         um[iz][iy] += u;
00140         vm[iz][iy] += v;
00141         wm[iz][iy] += w;
00142         pvm[iz][iy] += pv;
00143         h2om[iz][iy] += h2o;
00144         o3m[iz][iy] += o3;
00145         lwcm[iz][iy] += lwc;
00146         iwcm[iz][iy] += iwc;
00147         psm[iz][iy] += ps;
00148         tsm[iz][iy] += ts;
00149         zsm[iz][iy] += zs;
00150         usm[iz][iy] += us;
00151         vsm[iz][iy] += vs;
00152         pblm[iz][iy] += pbl;
00153         pctm[iz][iy] += pct;
00154         pcbm[iz][iy] += pcb;
00155         clm[iz][iy] += cl;
00156         if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00157             && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00158             plclm[iz][iy] += plcl;
00159             plfcm[iz][iy] += plfc;
00160             pelm[iz][iy] += pel;
00161             capem[iz][iy] += cape;
00162             cinm[iz][iy] += cin;
00163             npc[iz][iy]++;
00164         }
00165         if (gsl_finite(pt)) {
00166             ptm[iz][iy] += pt;
00167             ztm[iz][iy] += zt;
00168             ttm[iz][iy] += tt;
00169             h2otm[iz][iy] += h2ot;
00170             npt[iz][iy]++;
00171         }
00172         rhm[iz][iy] += RH(plev[iz], t, h2o);
00173         rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
00174         tdewm[iz][iy] += TDEW(plev[iz], h2o);
00175         ticem[iz][iy] += TICE(plev[iz], h2o);
00176         hno3m[iz][iy] += clim_hno3(met->time, lats[iy], plev[iz]);
00177         tnatm[iz][iy] +=
00178             nat_temperature(plev[iz], h2o,
00179                 clim_hno3(met->time, lats[iy], plev[iz]));
00180         ohm[iz][iy] += clim_oh(met->time, lats[iy], plev[iz]);
00181         np[iz][iy]++;
00182     }
00183 }
00184
00185 /* Create output file... */
00186 LOG(1, "Write meteorological data file: %s", argv[2]);
00187 if (!(out = fopen(argv[2], "w")))
00188     ERRMSG("Cannot create file!");
00189
00190 /* Write header... */
00191 fprintf(out,
00192     "# $1 = time [s]\n"
00193     "# $2 = altitude [km]\n"
00194     "# $3 = longitude [deg]\n"
00195     "# $4 = latitude [deg]\n"
00196     "# $5 = pressure [hPa]\n"
00197     "# $6 = temperature [K]\n"
00198     "# $7 = zonal wind [m/s]\n"
00199     "# $8 = meridional wind [m/s]\n"
00200     "# $9 = vertical velocity [hPa/s]\n"
00201     "# $10 = H2O volume mixing ratio [ppv]\n");
00202 fprintf(out,
00203     "# $11 = O3 volume mixing ratio [ppv]\n"
00204     "# $12 = geopotential height [km]\n"
00205     "# $13 = potential vorticity [PVU]\n"
00206     "# $14 = surface pressure [hPa]\n"
00207     "# $15 = surface temperature [K]\n"
00208     "# $16 = surface geopotential height [km]\n"
00209     "# $17 = surface zonal wind [m/s]\n"
00210     "# $18 = surface meridional wind [m/s]\n"
00211     "# $19 = tropopause pressure [hPa]\n"
00212     "# $20 = tropopause geopotential height [km]\n");

```

```

00213 fprintf(out,
00214     "# $21 = tropopause temperature [K]\n"
00215     "# $22 = tropopause water vapor [ppv]\n"
00216     "# $23 = cloud liquid water content [kg/kg]\n"
00217     "# $24 = cloud ice water content [kg/kg]\n"
00218     "# $25 = total column cloud water [kg/m^2]\n"
00219     "# $26 = cloud top pressure [hPa]\n"
00220     "# $27 = cloud bottom pressure [hPa]\n"
00221     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00222     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00223     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00224 fprintf(out,
00225     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00226     "# $32 = convective inhibition (CIN) [J/kg]\n"
00227     "# $33 = relative humidity over water [%]\n"
00228     "# $34 = relative humidity over ice [%]\n"
00229     "# $35 = dew point temperature [K]\n"
00230     "# $36 = frost point temperature [K]\n"
00231     "# $37 = NAT temperature [K]\n"
00232     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00233     "# $39 = OH concentration [molec/cm^3]\n"
00234     "# $40 = boundary layer pressure [hPa]\n");
00235 fprintf(out,
00236     "# $41 = number of data points\n"
00237     "# $42 = number of tropopause data points\n"
00238     "# $43 = number of CAPE data points\n");
00239
00240 /* Write data... */
00241 for (iz = 0; iz < nz; iz++) {
00242     fprintf(out, "\n");
00243     for (iy = 0; iy < ny; iy++)
00244         fprintf(out,
00245             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00246             "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00247             " %g %g %g %g %g %d %d %d\n",
00248             timem[iz][iy] / np[iz][iy], Z(plev[iz]),
00249             lonm[iz][iy] / np[iz][iy], lats[iy],
00250             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00251             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00252             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00253             zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00254             psm[iz][iy] / np[iz][iy], tsm[iz][iy] / np[iz][iy],
00255             zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
00256             vsm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00257             ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
00258             h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00259             iwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
00260             pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],
00261             plclm[iz][iy] / npc[iz][iy], plfcm[iz][iy] / npc[iz][iy],
00262             pelm[iz][iy] / npc[iz][iy], capem[iz][iy] / npc[iz][iy],
00263             cinm[iz][iy] / npc[iz][iy], rhm[iz][iy] / np[iz][iy],
00264             rhicem[iz][iy] / np[iz][iy], tdewm[iz][iy] / np[iz][iy],
00265             ticem[iz][iy] / np[iz][iy], tnatm[iz][iy] / np[iz][iy],
00266             hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
00267             pblm[iz][iy] / np[iz][iy], np[iz][iy],
00268             npt[iz][iy], npc[iz][iy]);
00269 }
00270
00271 /* Close file... */
00272 fclose(out);
00273
00274 /* Free... */
00275 free(met);
00276
00277 return EXIT_SUCCESS;
00278 }

```

5.37 sedi.c File Reference

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

Functions

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

5.37.1 Detailed Description

Calculate sedimentation velocity.

Definition in file [sedi.c](#).

5.37.2 Function Documentation

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

Definition at line 27 of file [sedi.c](#).

```

00029     {
00030
00031     double eta, p, T, r_p, rho, Re, rho_p, vs;
00032
00033     /* Check arguments... */
00034     if (argc < 5)
00035         ERRMSG("Give parameters: <p> <T> <r_p> <rho_p>");
00036
00037     /* Read arguments... */
00038     p = atof(argv[1]);
00039     T = atof(argv[2]);
00040     r_p = atof(argv[3]);
00041     rho_p = atof(argv[4]);
00042
00043     /* Calculate sedimentation velocity... */
00044     vs = sedi(p, T, r_p, rho_p);
00045
00046     /* Density of dry air [kg / m^3]... */
00047     rho = 100. * p / (RA * T);
00048
00049     /* Dynamic viscosity of air [kg / (m s)]... */
00050     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00051
00052     /* Particle Reynolds number... */
00053     Re = 2e-6 * r_p * vs * rho / eta;
00054
00055     /* Convert... */
00056     printf("%g %g\n", vs, Re);
00057
00058     return EXIT_SUCCESS;
00059 }

```

Here is the call graph for this function:



5.38 sedi.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) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     double eta, p, T, r_p, rho, Re, rho_p, vs;
00032
00033     /* Check arguments... */
00034     if (argc < 5)
00035         ERRMSG("Give parameters: <p> <T> <r_p> <rho_p>");
00036
00037     /* Read arguments... */
00038     p = atof(argv[1]);
00039     T = atof(argv[2]);
00040     r_p = atof(argv[3]);
00041     rho_p = atof(argv[4]);
00042
00043     /* Calculate sedimentation velocity... */
00044     vs = sedi(p, T, r_p, rho_p);
00045
00046     /* Density of dry air [kg / m^3]... */
00047     rho = 100. * p / (RA * T);
00048
00049     /* Dynamic viscosity of air [kg / (m s)]... */
00050     eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00051
00052     /* Particle Reynolds number... */
00053     Re = 2e-6 * r_p * vs * rho / eta;
00054
00055     /* Convert... */
00056     printf("%g %g\n", vs, Re);
00057
00058     return EXIT_SUCCESS;
00059 }

```

5.39 time2jsec.c File Reference

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

Functions

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

5.39.1 Detailed Description

Convert date to Julian seconds.

Definition in file [time2jsec.c](#).

5.39.2 Function Documentation

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

Definition at line 27 of file `time2jsec.c`.

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

Here is the call graph for this function:



5.40 time2jsec.c

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

```
00028     int argc,
00029     char *argv[] {
00030
00031     double jsec, remain;
00032
00033     int day, hour, min, mon, sec, year;
00034
00035     /* Check arguments... */
00036     if (argc < 8)
00037         ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039     /* Read arguments... */
00040     year = atoi(argv[1]);
00041     mon = atoi(argv[2]);
00042     day = atoi(argv[3]);
00043     hour = atoi(argv[4]);
00044     min = atoi(argv[5]);
00045     sec = atoi(argv[6]);
00046     remain = atof(argv[7]);
00047
00048     /* Convert... */
00049     time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
00050     printf("%.2f\n", jsec);
00051
00052     return EXIT_SUCCESS;
00053 }
```

5.41 tnat.c File Reference

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

Functions

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

5.41.1 Detailed Description

Calculate PSC temperatures.

Definition in file [tnat.c](#).

5.41.2 Function Documentation

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

Definition at line 31 of file [tnat.c](#).

```
00033     {
00034
00035     /* Check arguments... */
00036     if (argc < 3)
00037         ERRMSG("Give parameters: <p> <h2o> <hno3>");
00038
00039     /* Get variables... */
00040     double p = atof(argv[1]);
00041     double h2o = atof(argv[2]);
00042     double hno3 = atof(argv[3]);
00043
00044     /* Calculate T_ice and T_NAT... */
```

```

00045 double tice = TICE(p, h2o);
00046 double tnat = nat_temperature(p, h2o, hno3);
00047
00048 /* Write output... */
00049 printf("p= %g hPa\n", p);
00050 printf("q_H2O= %g ppv\n", h2o);
00051 printf("q_HNO3= %g ppv\n", hno3);
00052 printf("T_ice= %g K\n", tice);
00053 printf("T_NAT= %g K\n", tnat);
00054
00055 return EXIT_SUCCESS;
00056 }

```

Here is the call graph for this function:



5.42 tnat.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) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Main...
00029  ----- */
00030
00031 int main(
00032     int argc,
00033     char *argv[]) {
00034
00035     /* Check arguments... */
00036     if (argc < 3)
00037         ERRMSG("Give parameters: <p> <h2o> <hno3>");
00038
00039     /* Get variables... */
00040     double p = atof(argv[1]);
00041     double h2o = atof(argv[2]);
00042     double hno3 = atof(argv[3]);
00043
00044     /* Calculate T_ice and T_NAT... */
00045     double tice = TICE(p, h2o);
00046     double tnat = nat_temperature(p, h2o, hno3);
00047
00048     /* Write output... */
00049     printf("p= %g hPa\n", p);
00050     printf("q_H2O= %g ppv\n", h2o);
00051     printf("q_HNO3= %g ppv\n", hno3);
00052     printf("T_ice= %g K\n", tice);
00053     printf("T_NAT= %g K\n", tnat);
00054
00055     return EXIT_SUCCESS;
00056 }

```

5.43 trac.c File Reference

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

Functions

- void `module_advect_mp` (`met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt)
Calculate advection of air parcels (mipoint method).
- void `module_advect_rk` (`met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt)
Calculate advection of air parcels (Runge-Kutta).
- 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, `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, `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, `met_t` *met0, `met_t` *met1, `atm_t` *atm)
Interpolate meteorological data for air parcel positions.
- void `module_oh_chem` (`ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt)
Calculate OH chemistry.
- 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_timesteps` (`ctl_t` *ctl, `atm_t` *atm, 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[])

5.43.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file [trac.c](#).

5.43.2 Function Documentation

5.43.2.1 module_advect_mp() void module_advect_mp (

```

    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate advection of air parcels (mipoint method).

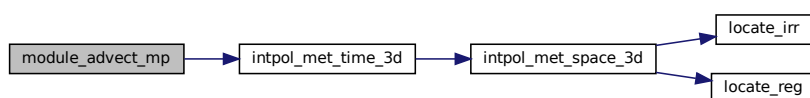
Definition at line 449 of file [trac.c](#).

```

00453     {
00454
00455     /* Set timer... */
00456     SELECT_TIMER("MODULE_ADEVECTION", "PHYSICS", NVTX_GPU);
00457
00458     const int np = atm->np;
00459     #ifndef _OPENACC
00460     #pragma acc data present(met0,met1,atm,dt)
00461     #pragma acc parallel loop independent gang vector
00462     #else
00463     #pragma omp parallel for default(shared)
00464     #endif
00465     for (int ip = 0; ip < np; ip++)
00466     if (dt[ip] != 0) {
00467
00468         double u, v, w;
00469
00470         /* Interpolate meteorological data... */
00471         INTPOL_INIT;
00472         INTPOL_3D(u, 1);
00473         INTPOL_3D(v, 0);
00474         INTPOL_3D(w, 0);
00475
00476         /* Get position of the mid point... */
00477         double dtm = atm->time[ip] + 0.5 * dt[ip];
00478         double xm0 =
00479             atm->lon[ip] + DX2DEG(0.5 * dt[ip] * u / 1000., atm->lat[ip]);
00480         double xml = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v / 1000.);
00481         double xm2 = atm->p[ip] + 0.5 * dt[ip] * w;
00482
00483         /* Interpolate meteorological data for mid point... */
00484         intpol_met_time_3d(met0, met0->u, met1, met1->u,
00485             dtm, xm2, xm0, xml, &u, ci, cw, 1);
00486         intpol_met_time_3d(met0, met0->v, met1, met1->v,
00487             dtm, xm2, xm0, xml, &v, ci, cw, 0);
00488         intpol_met_time_3d(met0, met0->w, met1, met1->w,
00489             dtm, xm2, xm0, xml, &w, ci, cw, 0);
00490
00491         /* Save new position... */
00492         atm->time[ip] += dt[ip];
00493         atm->lon[ip] += DX2DEG(dt[ip] * u / 1000., xml);
00494         atm->lat[ip] += DY2DEG(dt[ip] * v / 1000.);
00495         atm->p[ip] += dt[ip] * w;
00496     }
00497 }

```

Here is the call graph for this function:



```

5.43.2.2 module_advect_rk() void module_advect_rk (
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate advection of air parcels (Runge-Kutta).

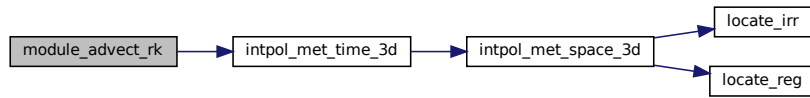
Definition at line 501 of file [trac.c](#).

```

00505     {
00506
00507     /* Set timer... */
00508     SELECT_TIMER("MODULE_ADVECTION", "PHYSICS", NVTX_GPU);
00509
00510     const int np = atm->np;
00511     #ifdef _OPENACC
00512     #pragma acc data present(met0,met1,atm,dt)
00513     #pragma acc parallel loop independent gang vector
00514     #else
00515     #pragma omp parallel for default(shared)
00516     #endif
00517     for (int ip = 0; ip < np; ip++)
00518     if (dt[ip] != 0) {
00519
00520         /* Init... */
00521         double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];
00522         INTPOL_INIT;
00523
00524         /* Loop over integration nodes... */
00525         for (int i = 0; i < 4; i++) {
00526
00527             /* Set position... */
00528             if (i == 0) {
00529                 dts = 0.0;
00530                 x[0] = atm->lon[ip];
00531                 x[1] = atm->lat[ip];
00532                 x[2] = atm->p[ip];
00533             } else {
00534                 dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
00535                 x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
00536                 x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
00537                 x[2] = atm->p[ip] + dts * w[i - 1];
00538             }
00539
00540             /* Interpolate meteo data... */
00541             double tm = atm->time[ip] + dts;
00542             intpol_met_time_3d(met0, met0->u, met1, met1->u, tm, x[2], x[0], x[1],
00543                             &u[i], ci, cw, 1);
00544             intpol_met_time_3d(met0, met0->v, met1, met1->v, tm, x[2], x[0], x[1],
00545                             &v[i], ci, cw, 0);
00546             intpol_met_time_3d(met0, met0->w, met1, met1->w, tm, x[2], x[0], x[1],
00547                             &w[i], ci, cw, 0);
00548
00549             /* Get mean wind... */
00550             double k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
00551             um += k * u[i];
00552             vm += k * v[i];
00553             wm += k * w[i];
00554         }
00555
00556         /* Set new position... */
00557         atm->time[ip] += dt[ip];
00558         atm->lon[ip] += DX2DEG(dt[ip] * um / 1000., atm->lat[ip]);
00559         atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
00560         atm->p[ip] += dt[ip] * wm;
00561     }
00562 }

```

Here is the call graph for this function:



5.43.2.3 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 566 of file trac.c.

```

00571     {
00572
00573     /* Set timer... */
00574     SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);
00575
00576     /* Check quantity flags... */
00577     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00578         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00579
00580     const int np = atm->np;
00581     #ifdef _OPENACC
00582     #pragma acc data present(ctl,met0,met1,atm,dt)
00583     #pragma acc parallel loop independent gang vector
00584     #else
00585     #pragma omp parallel for default(shared)
00586     #endif
00587     for (int ip = 0; ip < np; ip++)
00588     if (dt[ip] != 0) {
00589
00590         double ps;
00591
00592         /* Check latitude and pressure range... */
00593         if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
00594             || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
00595             continue;
00596
00597         /* Check surface layer... */
00598         if (ctl->bound_dps > 0) {
00599
00600             /* Get surface pressure... */
00601             INTPOL_INIT;
00602             INTPOL_2D(ps, 1);
00603
00604             /* Check whether particle is above the surface layer... */
00605             if (atm->p[ip] < ps - ctl->bound_dps)
00606                 continue;
00607         }
00608
00609         /* Set mass and volume mixing ratio... */
00610         if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
00611             atm->q[ctl->qnt_m][ip] = ctl->bound_mass;
00612         if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
00613             atm->q[ctl->qnt_vmr][ip] = ctl->bound_vmr;
00614     }
00615 }

```

```

5.43.2.4 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.

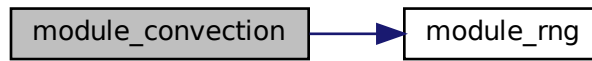
Definition at line 619 of file [trac.c](#).

```

00625         {
00626
00627         /* Set timer... */
00628         SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);
00629
00630         /* Create random numbers... */
00631         module_rng(rs, (size_t) atm->np, 0);
00632
00633         const int np = atm->np;
00634         #ifndef _OPENACC
00635         #pragma acc data present(ctl,met0,met1,atm,dt,rs)
00636         #pragma acc parallel loop independent gang vector
00637         #else
00638         #pragma omp parallel for default(shared)
00639         #endif
00640         for (int ip = 0; ip < np; ip++)
00641             if (dt[ip] != 0) {
00642
00643                 double cape, cin, pel, ps;
00644
00645                 /* Interpolate CAPE... */
00646                 INTPOL_INIT;
00647                 INTPOL_2D(cape, 1);
00648
00649                 /* Check threshold... */
00650                 if (isfinite(cape) && cape >= ctl->conv_cape) {
00651
00652                     /* Check CIN... */
00653                     if (ctl->conv_cin > 0) {
00654                         INTPOL_2D(cin, 0);
00655                         if (isfinite(cin) && cin >= ctl->conv_cin)
00656                             continue;
00657                     }
00658
00659                     /* Interpolate equilibrium level... */
00660                     INTPOL_2D(pel, 0);
00661
00662                     /* Check whether particle is above cloud top... */
00663                     if (!isfinite(pel) || atm->p[ip] < pel)
00664                         continue;
00665
00666                     /* Set pressure range for mixing... */
00667                     double pbot = atm->p[ip];
00668                     double ptop = atm->p[ip];
00669                     if (ctl->conv_mix_bot == 1) {
00670                         INTPOL_2D(ps, 0);
00671                         pbot = ps;
00672                     }
00673                     if (ctl->conv_mix_top == 1)
00674                         ptop = pel;
00675
00676                     /* Limit vertical velocity... */
00677                     if (ctl->conv_wmax > 0 || ctl->conv_wcape) {
00678                         double z = Z(atm->p[ip]);
00679                         double wmax = (ctl->conv_wcape) ? sqrt(2. * cape) : ctl->conv_wmax;
00680                         double pmax = P(z - wmax * dt[ip] / 1000.);
00681                         double pmin = P(z + wmax * dt[ip] / 1000.);
00682                         ptop = GSL_MAX(ptop, pmin);
00683                         pbot = GSL_MIN(pbot, pmax);
00684                     }
00685
00686                     /* Vertical mixing... */
00687                     atm->p[ip] = pbot + (ptop - pbot) * rs[ip];
00688                 }
00689             }
00690     }

```


Here is the call graph for this function:



5.43.2.5 module_decay() void module_decay (

```

    ctl_t * ctl,
    atm_t * atm,
    double * dt )
  
```

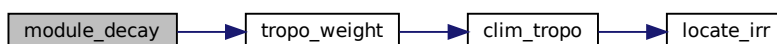
Calculate exponential decay of particle mass.

Definition at line 694 of file trac.c.

```

00697     {
00698
00699     /* Set timer... */
00700     SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);
00701
00702     /* Check quantity flags... */
00703     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00704         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00705
00706     const int np = atm->np;
00707     #ifndef _OPENACC
00708     #pragma acc data present(ctl,atm,dt)
00709     #pragma acc parallel loop independent gang vector
00710     #else
00711     #pragma omp parallel for default(shared)
00712     #endif
00713     for (int ip = 0; ip < np; ip++)
00714         if (dt[ip] != 0) {
00715
00716             /* Get weighting factor... */
00717             double w = tropo_weight(atm->time[ip], atm->lat[ip], atm->p[ip]);
00718
00719             /* Set lifetime... */
00720             double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00721
00722             /* Calculate exponential decay... */
00723             double aux = exp(-dt[ip] / tdec);
00724             if (ctl->qnt_m >= 0)
00725                 atm->q[ctl->qnt_m][ip] *= aux;
00726             if (ctl->qnt_vmr >= 0)
00727                 atm->q[ctl->qnt_vmr][ip] *= aux;
00728         }
00729 }
  
```

Here is the call graph for this function:



```

5.43.2.6 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 733 of file [trac.c](#).

```

00740     {
00741
00742     /* Set timer... */
00743     SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);
00744
00745     /* Create random numbers... */
00746     module_rng(rs, 3 * (size_t) atm->np, 1);
00747
00748     const int np = atm->np;
00749     #ifdef _OPENACC
00750     #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00751     #pragma acc parallel loop independent gang vector
00752     #else
00753     #pragma omp parallel for default(shared)
00754     #endif
00755     for (int ip = 0; ip < np; ip++)
00756     if (dt[ip] != 0) {
00757
00758     /* Get indices... */
00759     int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00760     int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00761     int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00762
00763     /* Caching of wind standard deviations... */
00764     if (cache->tsig[ix][iy][iz] != met0->time) {
00765
00766     /* Collect local wind data... */
00767     int n = 0;
00768     float u[16], v[16], w[16];
00769     for (int i = 0; i < 2; i++)
00770     for (int j = 0; j < 2; j++)
00771     for (int k = 0; k < 2; k++) {
00772     u[n] = met0->u[ix + i][iy + j][iz + k];
00773     v[n] = met0->v[ix + i][iy + j][iz + k];
00774     w[n] = met0->w[ix + i][iy + j][iz + k];
00775     n++;
00776     u[n] = met1->u[ix + i][iy + j][iz + k];
00777     v[n] = met1->v[ix + i][iy + j][iz + k];
00778     w[n] = met1->w[ix + i][iy + j][iz + k];
00779     n++;
00780     }
00781
00782     /* Get standard deviations of local wind data... */
00783     cache->uvwsig[ix][iy][iz][0] = stddev(u, n);
00784     cache->uvwsig[ix][iy][iz][1] = stddev(v, n);
00785     cache->uvwsig[ix][iy][iz][2] = stddev(w, n);
00786
00787     /* Save new time... */
00788     cache->tsig[ix][iy][iz] = met0->time;
00789     }
00790
00791     /* Set temporal correlations for mesoscale fluctuations... */
00792     double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00793     double r2 = sqrt(1 - r * r);
00794
00795     /* Calculate horizontal mesoscale wind fluctuations... */
00796     if (ctl->turb_mesox > 0) {
00797     cache->uvwp[ip][0] = (float)
00798     (r * cache->uvwp[ip][0]
00799     +
00800     r2 * rs[3 * ip] * ctl->turb_mesox * cache->uvwsig[ix][iy][iz][0]);
00801     atm->lon[ip] +=
00802     DX2DEG(cache->uvwp[ip][0] * dt[ip] / 1000., atm->lat[ip]);
00803
00804     cache->uvwp[ip][1] = (float)
00805     (r * cache->uvwp[ip][1]
00806     + r2 * rs[3 * ip +
00807     1] * ctl->turb_mesox * cache->uvwsig[ix][iy][iz][1]);
00808     atm->lat[ip] += DY2DEG(cache->uvwp[ip][1] * dt[ip] / 1000.);
00809     }

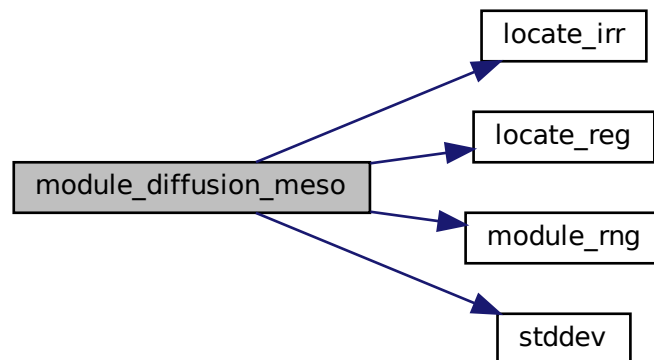
```

```

00810
00811     /* Calculate vertical mesoscale wind fluctuations... */
00812     if (ctl->turb_mesoz > 0) {
00813         cache->uvwp[ip][2] = (float)
00814             (r * cache->uvwp[ip][2]
00815              + r2 * rs[3 * ip +
00816                2] * ctl->turb_mesoz * cache->uvwsig[ix][iy][iz][2]);
00817         atm->p[ip] += cache->uvwp[ip][2] * dt[ip];
00818     }
00819 }
00820 }

```

Here is the call graph for this function:



5.43.2.7 module_diffusion_turb() void module_diffusion_turb (

```

    ctl_t * ctl,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate turbulent diffusion.

Definition at line 824 of file trac.c.

```

00828     {
00829
00830     /* Set timer... */
00831     SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);
00832
00833     /* Create random numbers... */
00834     module_rng(rs, 3 * (size_t) atm->np, 1);
00835
00836     const int np = atm->np;
00837     #ifdef _OPENACC
00838     #pragma acc data present(ctl,atm,dt,rs)
00839     #pragma acc parallel loop independent gang vector
00840     #else
00841     #pragma omp parallel for default(shared)
00842     #endif
00843     for (int ip = 0; ip < np; ip++)
00844         if (dt[ip] != 0) {
00845
00846             /* Get weighting factor... */
00847             double w = tropo_weight(atm->time[ip], atm->lat[ip], atm->p[ip]);
00848
00849             /* Set diffusivity... */

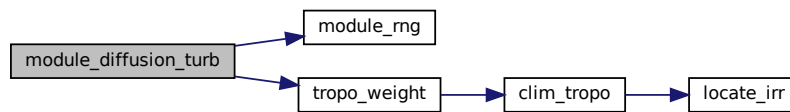
```

```

00850     double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00851     double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00852
00853     /* Horizontal turbulent diffusion... */
00854     if (dx > 0) {
00855         double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00856         atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00857         atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00858     }
00859
00860     /* Vertical turbulent diffusion... */
00861     if (dz > 0) {
00862         double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00863         atm->p[ip]
00864             += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00865     }
00866 }
00867 }

```

Here is the call graph for this function:



5.43.2.8 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 871 of file [trac.c](#).

```

00876     {
00877
00878     /* Set timer... */
00879     SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);
00880
00881     /* Width of the surface layer [hPa]. */
00882     const double dp = 30.;
00883
00884     /* Check quantity flags... */
00885     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00886         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00887
00888     const int np = atm->np;
00889     #ifdef _OPENACC
00890     #pragma acc data present(ctl,met0,met1,atm,dt)
00891     #pragma acc parallel loop independent gang vector
00892     #else
00893     #pragma omp parallel for default(shared)
00894     #endif
00895     for (int ip = 0; ip < np; ip++)
00896         if (dt[ip] != 0) {
00897
00898             double ps, t, v_dep;
00899
00900             /* Get surface pressure... */
00901             INTPOL_INIT;
00902             INTPOL_2D(ps, 1);
00903
00904             /* Check whether particle is above the surface layer... */

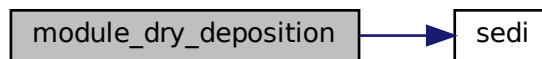
```

```

00905     if (atm->p[ip] < ps - dp)
00906         continue;
00907
00908     /* Set width of surface layer... */
00909     double dz = 1000. * (Z(ps - dp) - Z(ps));
00910
00911     /* Calculate sedimentation velocity for particles... */
00912     if (ctl->qnt_r > 0 && ctl->qnt_rho > 0) {
00913
00914         /* Get temperature... */
00915         INTPOL_3D(t, 1);
00916
00917         /* Set deposition velocity... */
00918         v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
00919                     atm->q[ctl->qnt_rho][ip]);
00920     }
00921
00922     /* Use explicit sedimentation velocity for gases... */
00923     else
00924         v_dep = ctl->dry_depo[0];
00925
00926     /* Calculate loss of mass based on deposition velocity... */
00927     double aux = exp(-dt[ip] * v_dep / dz);
00928     if (ctl->qnt_m >= 0)
00929         atm->q[ctl->qnt_m][ip] *= aux;
00930     if (ctl->qnt_vmr >= 0)
00931         atm->q[ctl->qnt_vmr][ip] *= aux;
00932 }
00933 }

```

Here is the call graph for this function:



5.43.2.9 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 937 of file [trac.c](#).

```

00942     {
00943
00944     FILE *in;
00945
00946     char line[LEN];
00947
00948     double t;
00949
00950     /* Set timer... */
00951     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
00952
00953     /* Init... */
00954     INTPOL_INIT;
00955
00956     /* Save pressure... */
00957     if (ctl->isosurf == 1)
00958         for (int ip = 0; ip < atm->np; ip++)

```

```

00959     cache->iso_var[ip] = atm->p[ip];
00960
00961     /* Save density... */
00962     else if (ctl->isosurf == 2)
00963         for (int ip = 0; ip < atm->np; ip++) {
00964             INTPOL_3D(t, 1);
00965             cache->iso_var[ip] = atm->p[ip] / t;
00966         }
00967
00968     /* Save potential temperature... */
00969     else if (ctl->isosurf == 3)
00970         for (int ip = 0; ip < atm->np; ip++) {
00971             INTPOL_3D(t, 1);
00972             cache->iso_var[ip] = THETA(atm->p[ip], t);
00973         }
00974
00975     /* Read balloon pressure data... */
00976     else if (ctl->isosurf == 4) {
00977
00978         /* Write info... */
00979         LOG(1, "Read balloon pressure data: %s", ctl->balloon);
00980
00981         /* Open file... */
00982         if (!(in = fopen(ctl->balloon, "r")))
00983             ERRMSG("Cannot open file!");
00984
00985         /* Read pressure time series... */
00986         while (fgets(line, LEN, in))
00987             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00988                     &(cache->iso_ps[cache->iso_n])) == 2)
00989                 if ((++cache->iso_n) > NP)
00990                     ERRMSG("Too many data points!");
00991
00992         /* Check number of points... */
00993         if (cache->iso_n < 1)
00994             ERRMSG("Could not read any data!");
00995
00996         /* Close file... */
00997         fclose(in);
00998     }
00999 }

```

5.43.2.10 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 1003 of file [trac.c](#).

```

01008     {
01009
01010         /* Set timer... */
01011         SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01012
01013         const int np = atm->np;
01014         #ifdef _OPENACC
01015         #pragma acc data present(ctl,met0,met1,atm,cache)
01016         #pragma acc parallel loop independent gang vector
01017         #else
01018         #pragma omp parallel for default(shared)
01019         #endif
01020         for (int ip = 0; ip < np; ip++) {
01021
01022             double t;
01023
01024             /* Init... */
01025             INTPOL_INIT;
01026
01027             /* Restore pressure... */
01028             if (ctl->isosurf == 1)
01029                 atm->p[ip] = cache->iso_var[ip];
01030
01031             /* Restore density... */
01032             else if (ctl->isosurf == 2) {

```

```

01033     INTPOL_3D(t, 1);
01034     atm->p[ip] = cache->iso_var[ip] * t;
01035 }
01036
01037 /* Restore potential temperature... */
01038 else if (ctl->isosurf == 3) {
01039     INTPOL_3D(t, 1);
01040     atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
01041 }
01042
01043 /* Interpolate pressure... */
01044 else if (ctl->isosurf == 4) {
01045     if (atm->time[ip] <= cache->iso_ts[0])
01046         atm->p[ip] = cache->iso_ps[0];
01047     else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
01048         atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
01049     else {
01050         int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
01051         atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
01052                        cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
01053                        atm->time[ip]);
01054     }
01055 }
01056 }
01057 }

```

Here is the call graph for this function:



5.43.2.11 module_meteo() void module_meteo (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm )

```

Interpolate meteorological data for air parcel positions.

Definition at line 1061 of file trac.c.

```

01065     {
01066
01067     /* Set timer... */
01068     SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);
01069
01070     /* Check quantity flags... */
01071     if (ctl->qnt_tsts >= 0)
01072         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01073             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01074
01075     const int np = atm->np;
01076     #ifdef _OPENACC
01077     #pragma acc data present(ctl,met0,met1,atm)
01078     #pragma acc parallel loop independent gang vector
01079     #else
01080     #pragma omp parallel for default(shared)
01081     #endif
01082     for (int ip = 0; ip < np; ip++) {
01083
01084         double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
01085                cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
01086
01087     /* Interpolate meteorological data... */

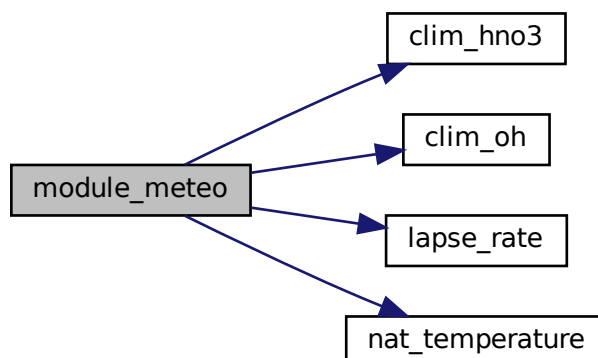
```

```

01088     INTPOL_INIT;
01089     INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
01090
01091     /* Set quantities... */
01092     SET_ATM(qnt_ps, ps);
01093     SET_ATM(qnt_ts, ts);
01094     SET_ATM(qnt_zs, zs);
01095     SET_ATM(qnt_us, us);
01096     SET_ATM(qnt_vs, vs);
01097     SET_ATM(qnt_pbl, pbl);
01098     SET_ATM(qnt_pt, pt);
01099     SET_ATM(qnt_tt, tt);
01100     SET_ATM(qnt_zt, zt);
01101     SET_ATM(qnt_h2ot, h2ot);
01102     SET_ATM(qnt_p, atm->p[ip]);
01103     SET_ATM(qnt_z, z);
01104     SET_ATM(qnt_t, t);
01105     SET_ATM(qnt_u, u);
01106     SET_ATM(qnt_v, v);
01107     SET_ATM(qnt_w, w);
01108     SET_ATM(qnt_h2o, h2o);
01109     SET_ATM(qnt_o3, o3);
01110     SET_ATM(qnt_lwc, lwc);
01111     SET_ATM(qnt_iwc, iwc);
01112     SET_ATM(qnt_pct, pct);
01113     SET_ATM(qnt_pcb, pcb);
01114     SET_ATM(qnt_cl, cl);
01115     SET_ATM(qnt_plcl, plcl);
01116     SET_ATM(qnt_plfc, plfc);
01117     SET_ATM(qnt_pel, pel);
01118     SET_ATM(qnt_cape, cape);
01119     SET_ATM(qnt_cin, cin);
01120     SET_ATM(qnt_hno3, clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]));
01121     SET_ATM(qnt_oh, clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01122     SET_ATM(qnt_vh, sqrt(u * u + v * v));
01123     SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01124     SET_ATM(qnt_psat, PSAT(t));
01125     SET_ATM(qnt_psice, PSICE(t));
01126     SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
01127     SET_ATM(qnt_sh, SH(h2o));
01128     SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
01129     SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01130     SET_ATM(qnt_theta, THETA(atm->p[ip], t));
01131     SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
01132     SET_ATM(qnt_tvirt, TVIRT(t, h2o));
01133     SET_ATM(qnt_lapse, lapse_rate(t, h2o));
01134     SET_ATM(qnt_pv, pv);
01135     SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
01136     SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
01137     SET_ATM(qnt_tnat,
01138             nat_temperature(atm->p[ip], h2o,
01139                     clim_hno3(atm->time[ip], atm->lat[ip],
01140                             atm->p[ip])));
01141     SET_ATM(qnt_tsts,
01142             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01143 }
01144 }

```


Here is the call graph for this function:



5.43.2.12 module_oh_chem() void module_oh_chem (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate OH chemistry.

Definition at line 1148 of file [trac.c](#).

```

01153     {
01154
01155     /* Set timer... */
01156     SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);
01157
01158     /* Check quantity flags... */
01159     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01160         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01161
01162     const int np = atm->np;
01163     #ifndef _OPENACC
01164     #pragma acc data present(ctl,met0,met1,atm,dt)
01165     #pragma acc parallel loop independent gang vector
01166     #else
01167     #pragma omp parallel for default(shared)
01168     #endif
01169     for (int ip = 0; ip < np; ip++)
01170         if (dt[ip] != 0) {
01171
01172             /* Get temperature... */
01173             double t;
01174             INTPOL_INIT;
01175             INTPOL_3D(t, 1);
01176
01177             /* Use constant reaction rate... */
01178             double k = GSL_NAN;
01179             if (ctl->oh_chem_reaction == 1)
01180                 k = ctl->oh_chem[0];
01181
01182             /* Calculate bimolecular reaction rate... */
01183             else if (ctl->oh_chem_reaction == 2)
01184                 k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);
01185
01186             /* Calculate termolecular reaction rate... */

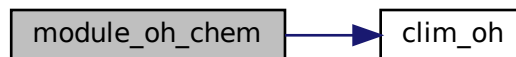
```

```

01187     if (ctl->oh_chem_reaction == 3) {
01188
01189         /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
01190         double M = 7.243e21 * (atm->p[ip] / 1000.) / t;
01191
01192         /* Calculate rate coefficient for X + OH + M -> XOH + M
01193            (JPL Publication 19-05) ... */
01194         double k0 = ctl->oh_chem[0] *
01195             (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
01196         double ki = ctl->oh_chem[2] *
01197             (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
01198         double c = log10(k0 * M / ki);
01199         k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01200     }
01201
01202     /* Calculate exponential decay... */
01203     double aux
01204         = exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01205     if (ctl->qnt_m >= 0)
01206         atm->q[ctl->qnt_m][ip] *= aux;
01207     if (ctl->qnt_vmr >= 0)
01208         atm->q[ctl->qnt_vmr][ip] *= aux;
01209 }
01210 }

```

Here is the call graph for this function:



5.43.2.13 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 1214 of file [trac.c](#).

```

01219     {
01220
01221         /* Set timer... */
01222         SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);
01223
01224         const int np = atm->np;
01225         #ifdef _OPENACC
01226         #pragma acc data present(met0,met1,atm,dt)
01227         #pragma acc parallel loop independent gang vector
01228         #else
01229         #pragma omp parallel for default(shared)
01230         #endif
01231         for (int ip = 0; ip < np; ip++)
01232             if (dt[ip] != 0) {
01233
01234                 /* Init... */
01235                 double ps;
01236                 INTPOL_INIT;
01237
01238                 /* Calculate modulo... */
01239                 atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01240                 atm->lat[ip] = FMOD(atm->lat[ip], 360.);

```

```

01241
01242     /* Check latitude... */
01243     while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01244         if (atm->lat[ip] > 90) {
01245             atm->lat[ip] = 180 - atm->lat[ip];
01246             atm->lon[ip] += 180;
01247         }
01248         if (atm->lat[ip] < -90) {
01249             atm->lat[ip] = -180 - atm->lat[ip];
01250             atm->lon[ip] += 180;
01251         }
01252     }
01253
01254     /* Check longitude... */
01255     while (atm->lon[ip] < -180)
01256         atm->lon[ip] += 360;
01257     while (atm->lon[ip] >= 180)
01258         atm->lon[ip] -= 360;
01259
01260     /* Check pressure... */
01261     if (atm->p[ip] < met0->p[met0->np - 1]) {
01262         if (ctl->reflect)
01263             atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
01264         else
01265             atm->p[ip] = met0->p[met0->np - 1];
01266     } else if (atm->p[ip] > 300.) {
01267         INTPOL_2D(ps, 1);
01268         if (atm->p[ip] > ps) {
01269             if (ctl->reflect)
01270                 atm->p[ip] = 2. * ps - atm->p[ip];
01271             else
01272                 atm->p[ip] = ps;
01273         }
01274     }
01275 }
01276 }

```

5.43.2.14 module_rng_init() void module_rng_init (int ntask)

Initialize random number generator...

Definition at line 1280 of file trac.c.

```

01281     {
01282
01283         /* Initialize random number generator... */
01284         #ifdef _OPENACC
01285
01286         if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
01287             != CURAND_STATUS_SUCCESS)
01288             ERRMSG("Cannot create random number generator!");
01289         if (curandSetPseudoRandomGeneratorSeed(rng, ntask)
01290             != CURAND_STATUS_SUCCESS)
01291             ERRMSG("Cannot set seed for random number generator!");
01292         if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01293             != CURAND_STATUS_SUCCESS)
01294             ERRMSG("Cannot set stream for random number generator!");
01295
01296         #else
01297         gsl_rng_env_setup();
01298         if (omp_get_max_threads() > NTHREADS)
01299             ERRMSG("Too many threads!");
01300         for (int i = 0; i < NTHREADS; i++) {
01301             rng[i] = gsl_rng_alloc(gsl_rng_default);
01302             gsl_rng_set(rng[i], gsl_rng_default_seed
01303                 + (long unsigned) (ntask * NTHREADS + i));
01304         }
01305     }
01306
01307     #endif
01308 }

```

```

5.43.2.15 module_rng() void module_rng (
    double * rs,
    size_t n,
    int method )

```

Generate random numbers.

Definition at line 1312 of file [trac.c](#).

```

01315     {
01316
01317     #ifdef _OPENACC
01318
01319     #pragma acc host_data use_device(rs)
01320     {
01321         /* Uniform distribution... */
01322         if (method == 0) {
01323             if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n))
01324                 != CURAND_STATUS_SUCCESS)
01325                 ERRMSG("Cannot create random numbers!");
01326         }
01327
01328         /* Normal distribution... */
01329         else if (method == 1) {
01330             if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0)
01331                 != CURAND_STATUS_SUCCESS)
01332                 ERRMSG("Cannot create random numbers!");
01333         }
01334     }
01335
01336     #else
01337
01338     /* Uniform distribution... */
01339     if (method == 0) {
01340         #pragma omp parallel for default(shared)
01341         for (size_t i = 0; i < n; ++i)
01342             rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01343     }
01344
01345     /* Normal distribution... */
01346     else if (method == 1) {
01347         #pragma omp parallel for default(shared)
01348         for (size_t i = 0; i < n; ++i)
01349             rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01350     }
01351     #endif
01352
01353 }

```

```

5.43.2.16 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 1357 of file [trac.c](#).

```

01362     {
01363
01364     /* Set timer... */
01365     SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);
01366
01367     const int np = atm->np;
01368     #ifdef _OPENACC
01369     #pragma acc data present(ctl,met0,met1,atm,dt)
01370     #pragma acc parallel loop independent gang vector
01371     #else
01372     #pragma omp parallel for default(shared)
01373     #endif
01374     for (int ip = 0; ip < np; ip++)
01375         if (dt[ip] != 0) {
01376
01377         /* Get temperature... */

```

```

01378     double t;
01379     INTPOL_INIT;
01380     INTPOL_3D(t, 1);
01381
01382     /* Sedimentation velocity... */
01383     double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
01384                      atm->q[ctl->qnt_rho][ip]);
01385
01386     /* Calculate pressure change... */
01387     atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01388 }
01389 }

```

Here is the call graph for this function:



5.43.2.17 module_timesteps() void module_timesteps (

```

    ctl_t * ctl,
    atm_t * atm,
    double * dt,
    double t )

```

Calculate time steps.

Definition at line 1393 of file trac.c.

```

01397     {
01398
01399     /* Set timer... */
01400     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01401
01402     const int np = atm->np;
01403     #ifdef _OPENACC
01404     #pragma acc data present(ctl,atm,dt)
01405     #pragma acc parallel loop independent gang vector
01406     #else
01407     #pragma omp parallel for default(shared)
01408     #endif
01409     for (int ip = 0; ip < np; ip++) {
01410         if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
01411             && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
01412             && ctl->direction * (atm->time[ip] - t) < 0))
01413             dt[ip] = t - atm->time[ip];
01414         else
01415             dt[ip] = 0.0;
01416     }
01417 }

```

5.43.2.18 module_timesteps_init() void module_timesteps_init (

```

    ctl_t * ctl,
    atm_t * atm )

```

Initialize timesteps.

Definition at line 1421 of file [trac.c](#).

```

1423     {
1424
1425     /* Set timer... */
1426     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
1427
1428     /* Set start time... */
1429     if (ctl->direction == 1) {
1430         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
1431         if (ctl->t_stop > 1e99)
1432             ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
1433     } else {
1434         ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
1435         if (ctl->t_stop > 1e99)
1436             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
1437     }
1438
1439     /* Check time interval... */
1440     if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
1441         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
1442
1443     /* Round start time... */
1444     if (ctl->direction == 1)
1445         ctl->t_start = floor(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
1446     else
1447         ctl->t_start = ceil(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
1448 }

```

5.43.2.19 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 1452 of file [trac.c](#).

```

1457     {
1458
1459     /* Set timer... */
1460     SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);
1461
1462     /* Check quantity flags... */
1463     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
1464         ERRMSG("Module needs quantity mass or volume mixing ratio!");
1465
1466     const int np = atm->np;
1467     #ifdef _OPENACC
1468     #pragma acc data present(ctl,met0,met1,atm,dt)
1469     #pragma acc parallel loop independent gang vector
1470     #else
1471     #pragma omp parallel for default(shared)
1472     #endif
1473     for (int ip = 0; ip < np; ip++)
1474         if (dt[ip] != 0) {
1475
1476             double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;
1477
1478             int inside;
1479
1480             /* Check whether particle is below cloud top... */
1481             INTPOL_INIT;
1482             INTPOL_2D(pct, 1);
1483             if (!isfinite(pct) || atm->p[ip] <= pct)
1484                 continue;
1485
1486             /* Get cloud bottom pressure... */

```

```

01487     INTPOL_2D(pcb, 0);
01488
01489     /* Estimate precipitation rate (Pisso et al., 2019)... */
01490     INTPOL_2D(cl, 0);
01491     double Is = pow(2. * cl, 1. / 0.36);
01492     if (Is < 0.01)
01493         continue;
01494
01495     /* Check whether particle is inside or below cloud... */
01496     INTPOL_3D(lwc, 1);
01497     INTPOL_3D(iwc, 0);
01498     inside = (iwc > 0 || lwc > 0);
01499
01500     /* Calculate in-cloud scavenging coefficient... */
01501     if (inside) {
01502
01503         /* Use exponential dependency for particles... */
01504         if (ctl->wet_depo[0] > 0)
01505             lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01506
01507         /* Use Henry's law for gases... */
01508         else if (ctl->wet_depo[2] > 0) {
01509
01510             /* Get temperature... */
01511             INTPOL_3D(t, 0);
01512
01513             /* Get Henry's constant (Sander, 2015)... */
01514             h = ctl->wet_depo[2]
01515                 * exp(ctl->wet_depo[3] * (1. / t - 1. / 298.15));
01516
01517             /* Estimate depth of cloud layer... */
01518             dz = 1e3 * (Z(pct) - Z(pcb));
01519
01520             /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01521             lambda = h * RI * t * Is / 3.6e6 / dz;
01522         }
01523     }
01524
01525     /* Calculate below-cloud scavenging coefficient... */
01526     else {
01527
01528         /* Use exponential dependency for particles... */
01529         if (ctl->wet_depo[4] > 0)
01530             lambda = ctl->wet_depo[4] * pow(Is, ctl->wet_depo[5]);
01531
01532         /* Use Henry's law for gases... */
01533         else if (ctl->wet_depo[6] > 0) {
01534
01535             /* Get temperature... */
01536             INTPOL_3D(t, 0);
01537
01538             /* Get Henry's constant (Sander, 2015)... */
01539             h = ctl->wet_depo[6]
01540                 * exp(ctl->wet_depo[7] * (1. / t - 1. / 298.15));
01541
01542             /* Estimate depth of cloud layer... */
01543             dz = 1e3 * (Z(pct) - Z(pcb));
01544
01545             /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01546             lambda = h * RI * t * Is / 3.6e6 / dz;
01547         }
01548     }
01549
01550     /* Calculate exponential decay of mass... */
01551     double aux = exp(-dt[ip] * lambda);
01552     if (ctl->qnt_m >= 0)
01553         atm->q[ctl->qnt_m][ip] *= aux;
01554     if (ctl->qnt_vmr >= 0)
01555         atm->q[ctl->qnt_vmr][ip] *= aux;
01556 }
01557 }

```

5.43.2.20 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.

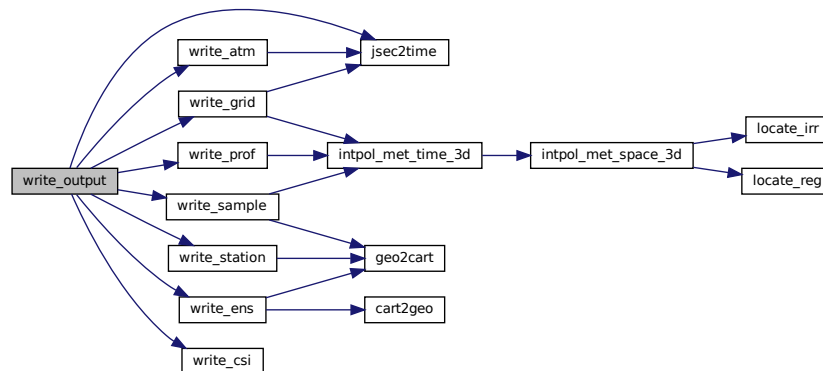
Definition at line 1561 of file [trac.c](#).

```

01567     {
01568
01569     char filename[2 * LEN];
01570
01571     double r;
01572
01573     int year, mon, day, hour, min, sec;
01574
01575     /* Get time... */
01576     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01577
01578     /* Update host... */
01579 #ifdef _OPENACC
01580     if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
01581         || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
01582         || ctl->csi_basename[0] != '-' || ctl->ens_basename[0] != '-'
01583         || ctl->prof_basename[0] != '-' || ctl->sample_basename[0] != '-'
01584         || ctl->stat_basename[0] != '-') {
01585         SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
01586 #pragma acc update host(atm[:1])
01587     }
01588 #endif
01589
01590     /* Write atmospheric data... */
01591     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01592         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01593             dirname, ctl->atm_basename, year, mon, day, hour, min);
01594         write_atm(filename, ctl, atm, t);
01595     }
01596
01597     /* Write gridded data... */
01598     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01599         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01600             dirname, ctl->grid_basename, year, mon, day, hour, min);
01601         write_grid(filename, ctl, met0, met1, atm, t);
01602     }
01603
01604     /* Write CSI data... */
01605     if (ctl->csi_basename[0] != '-') {
01606         sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01607         write_csi(filename, ctl, atm, t);
01608     }
01609
01610     /* Write ensemble data... */
01611     if (ctl->ens_basename[0] != '-') {
01612         sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01613         write_ens(filename, ctl, atm, t);
01614     }
01615
01616     /* Write profile data... */
01617     if (ctl->prof_basename[0] != '-') {
01618         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01619         write_prof(filename, ctl, met0, met1, atm, t);
01620     }
01621
01622     /* Write sample data... */
01623     if (ctl->sample_basename[0] != '-') {
01624         sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
01625         write_sample(filename, ctl, met0, met1, atm, t);
01626     }
01627
01628     /* Write station data... */
01629     if (ctl->stat_basename[0] != '-') {
01630         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01631         write_station(filename, ctl, atm, t);
01632     }
01633 }

```


Here is the call graph for this function:



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

Definition at line 193 of file [trac.c](#).

```

00195     {
00196
00197     ctl_t ctl;
00198
00199     atm_t *atm;
00200
00201     cache_t *cache;
00202
00203     met_t *met0, *met1;
00204
00205     FILE *dirlist;
00206
00207     char dirname[LEN], filename[2 * LEN];
00208
00209     double *dt, *rs, t;
00210
00211     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00212
00213     /* Start timers... */
00214     START_TIMERS;
00215
00216     /* Initialize MPI... */
00217 #ifdef MPI
00218     SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
00219     MPI_Init(&argc, &argv);
00220     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00221     MPI_Comm_size(MPI_COMM_WORLD, &size);
00222 #endif
00223
00224     /* Initialize GPUs... */
00225 #ifdef _OPENACC
00226     SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
00227     num_devices = acc_get_num_devices(acc_device_nvidia);
00228     if (num_devices <= 0)
00229         ERRMSG("Not running on a GPU device!");
00230     int device_num = rank % num_devices;
00231     acc_set_device_num(device_num, acc_device_nvidia);
00232     acc_device_t device_type = acc_get_device_type();
00233     acc_init(device_type);
00234 #endif
00235
00236     /* Check arguments... */
00237     if (argc < 4)
00238         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00239
00240     /* Open directory list... */
  
```

```

00241  if (!(dirlist = fopen(argv[1], "r")))
00242      ERRMSG("Cannot open directory list!");
00243
00244  /* Loop over directories... */
00245  while (fscanf(dirlist, "%4999s", dirname) != EOF) {
00246
00247      /* MPI parallelization... */
00248      if ((++ntask) % size != rank)
00249          continue;
00250
00251      /* -----
00252      Initialize model run...
00253      ----- */
00254
00255      /* Allocate... */
00256      SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
00257      ALLOC(atm, atm_t, 1);
00258      ALLOC(cache, cache_t, 1);
00259      ALLOC(met0, met_t, 1);
00260      ALLOC(met1, met_t, 1);
00261      ALLOC(dt, double,
00262            NP);
00263      ALLOC(rs, double,
00264            3 * NP + 1);
00265
00266      /* Create data region on GPUs... */
00267      #ifdef _OPENACC
00268          SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
00269          #pragma acc enter data create(atm[:1],cache[:1],ctl,met0[:1],met1[:1],dt[:NP],rs[:3*NP])
00270      #endif
00271
00272      /* Read control parameters... */
00273      sprintf(filename, "%s/%s", dirname, argv[2]);
00274      read_ctl(filename, argc, argv, &ctl);
00275
00276      /* Read atmospheric data... */
00277      sprintf(filename, "%s/%s", dirname, argv[3]);
00278      if (!read_atm(filename, &ctl, atm))
00279          ERRMSG("Cannot open file!");
00280
00281      /* Initialize timesteps... */
00282      module_timesteps_init(&ctl, atm);
00283
00284      /* Update GPU... */
00285      #ifdef _OPENACC
00286          SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00287          #pragma acc update device(atm[:1],ctl)
00288      #endif
00289
00290      /* Initialize random number generator... */
00291      module_rng_init(ntask);
00292
00293      /* Initialize meteorological data... */
00294      get_met(&ctl, ctl.t_start, &met0, &met1);
00295      if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00296          WARN("Violation of CFL criterion! Check DT_MOD!");
00297
00298      /* Initialize isosurface... */
00299      if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00300          module_isosurf_init(&ctl, met0, met1, atm, cache);
00301
00302      /* Update GPU... */
00303      #ifdef _OPENACC
00304          SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00305          #pragma acc update device(cache[:1])
00306      #endif
00307
00308      /* -----
00309      Loop over timesteps...
00310      ----- */
00311
00312      /* Loop over timesteps... */
00313      for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00314           t += ctl.direction * ctl.dt_mod) {
00315
00316          /* Adjust length of final time step... */
00317          if (ctl.direction * (t - ctl.t_stop) > 0)
00318              t = ctl.t_stop;
00319
00320          /* Set time steps of air parcels... */
00321          module_timesteps(&ctl, atm, dt, t);
00322
00323          /* Get meteorological data... */
00324          if (t != ctl.t_start)
00325              get_met(&ctl, t, &met0, &met1);
00326
00327          /* Check initial positions... */

```

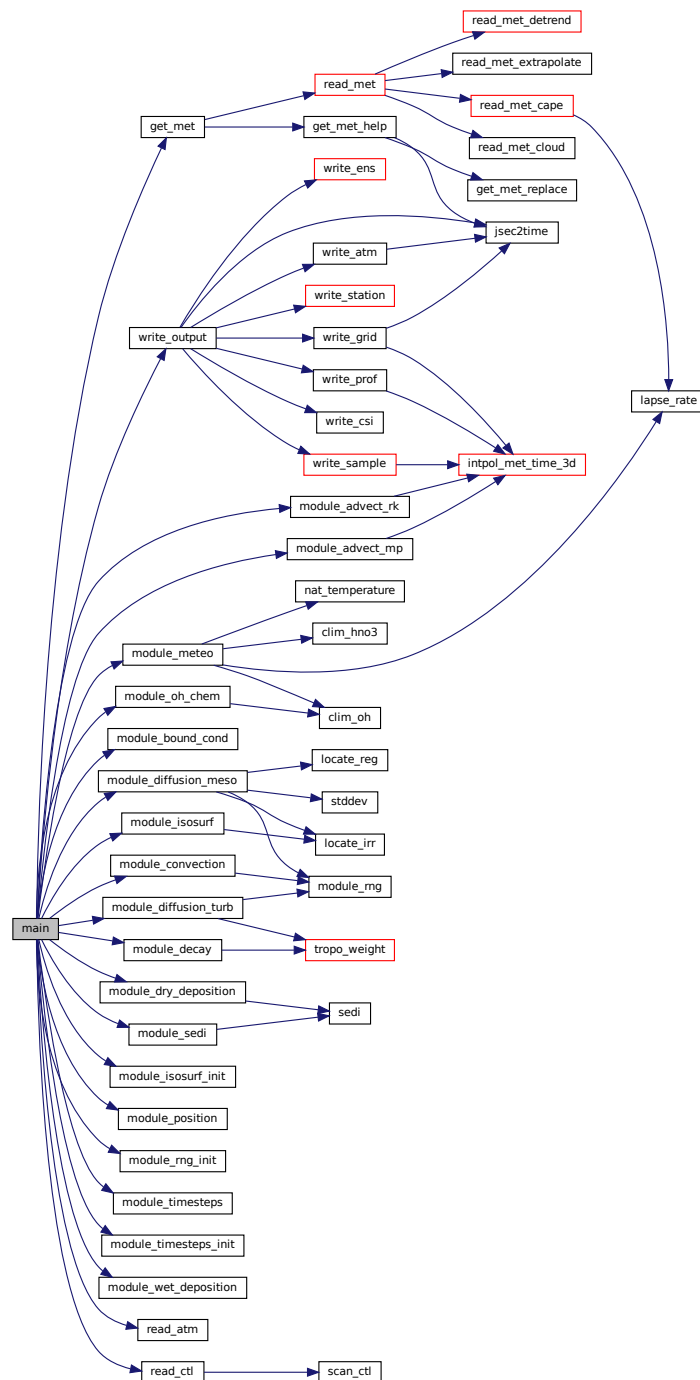
```

00328     module_position(&ctl, met0, met1, atm, dt);
00329
00330     /* Advection... */
00331     if (ctl.advect == 0)
00332         module_advect_mp(met0, met1, atm, dt);
00333     else
00334         module_advect_rk(met0, met1, atm, dt);
00335
00336     /* Turbulent diffusion... */
00337     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00338         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
00339         module_diffusion_turb(&ctl, atm, dt, rs);
00340
00341     /* Mesoscale diffusion... */
00342     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
00343         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00344
00345     /* Convection... */
00346     if (ctl.conv_cape >= 0
00347         && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
00348         module_convection(&ctl, met0, met1, atm, dt, rs);
00349
00350     /* Sedimentation... */
00351     if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00352         module_sedi(&ctl, met0, met1, atm, dt);
00353
00354     /* Isosurface... */
00355     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00356         module_isosurf(&ctl, met0, met1, atm, cache);
00357
00358     /* Check final positions... */
00359     module_position(&ctl, met0, met1, atm, dt);
00360
00361     /* Interpolate meteorological data... */
00362     if (ctl.met_dt_out > 0
00363         && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0))
00364         module_meteo(&ctl, met0, met1, atm);
00365
00366     /* Decay of particle mass... */
00367     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00368         module_decay(&ctl, atm, dt);
00369
00370     /* OH chemistry... */
00371     if (ctl.oh_chem_reaction != 0)
00372         module_oh_chem(&ctl, met0, met1, atm, dt);
00373
00374     /* Dry deposition... */
00375     if (ctl.dry_depo[0] > 0)
00376         module_dry_deposition(&ctl, met0, met1, atm, dt);
00377
00378     /* Wet deposition... */
00379     if ((ctl.wet_depo[0] > 0 || ctl.wet_depo[2] > 0)
00380         && (ctl.wet_depo[4] > 0 || ctl.wet_depo[6] > 0))
00381         module_wet_deposition(&ctl, met0, met1, atm, dt);
00382
00383     /* Boundary conditions... */
00384     if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
00385         module_bound_cond(&ctl, met0, met1, atm, dt);
00386
00387     /* Write output... */
00388     write_output(dirname, &ctl, met0, met1, atm, t);
00389 }
00390
00391 /* -----
00392    Finalize model run...
00393    ----- */
00394
00395 /* Report problem size... */
00396 LOG(1, "SIZE_NP = %d", atm->np);
00397 LOG(1, "SIZE_MPI_TASKS = %d", size);
00398 LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
00399 LOG(1, "SIZE_ACC_DEVICES = %d", num_devices);
00400
00401 /* Report memory usage... */
00402 LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
00403 LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
00404 LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
00405 LOG(1, "MEMORY_DYNAMIC = %g MByte", (sizeof(met_t)
00406                                     + 4 * NP * sizeof(double)
00407                                     + EX * EY * EP * sizeof(float)) /
00408     1024. / 1024.);
00409 LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * sizeof(double)
00410                                     + EX * EY * EP * sizeof(float)
00411                                     + 4 * GX * GY * GZ * sizeof(double)
00412                                     + 2 * GX * GY * GZ * sizeof(int)
00413                                     + 2 * GX * GY * sizeof(double)
00414                                     + GX * GY * sizeof(int)) / 1024. /

```

```
00415         1024.);
00416
00417     /* Delete data region on GPUs... */
00418 #ifdef _OPENACC
00419     SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
00420 #pragma acc exit data delete(ctl,atm,cache,met0,met1,dt,rs)
00421 #endif
00422
00423     /* Free... */
00424     SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
00425     free(atm);
00426     free(cache);
00427     free(met0);
00428     free(met1);
00429     free(dt);
00430     free(rs);
00431
00432     /* Report timers... */
00433     PRINT_TIMERS;
00434 }
00435
00436 /* Finalize MPI... */
00437 #ifdef MPI
00438     MPI_Finalize();
00439 #endif
00440
00441 /* Stop timers... */
00442 STOP_TIMERS;
00443
00444 return EXIT_SUCCESS;
00445 }
```

Here is the call graph for this function:



5.44 trac.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008

```

```
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028    Global variables...
00029    ----- */
00030
00031 #ifdef _OPENACC
00032 curandGenerator_t rng;
00033 #else
00034 static gsl_rng *rng[NTHREADS];
00035 #endif
00036
00037 /* -----
00038    Functions...
00039    ----- */
00040
00042 void module_advect_mp(
00043     met_t * met0,
00044     met_t * met1,
00045     atm_t * atm,
00046     double *dt);
00047
00049 void module_advect_rk(
00050     met_t * met0,
00051     met_t * met1,
00052     atm_t * atm,
00053     double *dt);
00054
00056 void module_bound_cond(
00057     ctl_t * ctl,
00058     met_t * met0,
00059     met_t * met1,
00060     atm_t * atm,
00061     double *dt);
00062
00064 void module_convection(
00065     ctl_t * ctl,
00066     met_t * met0,
00067     met_t * met1,
00068     atm_t * atm,
00069     double *dt,
00070     double *rs);
00071
00073 void module_decay(
00074     ctl_t * ctl,
00075     atm_t * atm,
00076     double *dt);
00077
00079 void module_diffusion_meso(
00080     ctl_t * ctl,
00081     met_t * met0,
00082     met_t * met1,
00083     atm_t * atm,
00084     cache_t * cache,
00085     double *dt,
00086     double *rs);
00087
00089 void module_diffusion_turb(
00090     ctl_t * ctl,
00091     atm_t * atm,
00092     double *dt,
00093     double *rs);
00094
00096 void module_dry_deposition(
00097     ctl_t * ctl,
00098     met_t * met0,
00099     met_t * met1,
00100     atm_t * atm,
00101     double *dt);
00102
00104 void module_isosurf_init(
00105     ctl_t * ctl,
00106     met_t * met0,
00107     met_t * met1,
00108     atm_t * atm,
00109     cache_t * cache);
```

```

00110
00112 void module_isosurf(
00113     ctl_t * ctl,
00114     met_t * met0,
00115     met_t * met1,
00116     atm_t * atm,
00117     cache_t * cache);
00118
00120 void module_meteo(
00121     ctl_t * ctl,
00122     met_t * met0,
00123     met_t * met1,
00124     atm_t * atm);
00125
00127 void module_oh_chem(
00128     ctl_t * ctl,
00129     met_t * met0,
00130     met_t * met1,
00131     atm_t * atm,
00132     double *dt);
00133
00135 void module_position(
00136     ctl_t * ctl,
00137     met_t * met0,
00138     met_t * met1,
00139     atm_t * atm,
00140     double *dt);
00141
00143 void module_rng_init(
00144     int ntask);
00145
00147 void module_rng(
00148     double *rs,
00149     size_t n,
00150     int method);
00151
00153 void module_sedi(
00154     ctl_t * ctl,
00155     met_t * met0,
00156     met_t * met1,
00157     atm_t * atm,
00158     double *dt);
00159
00161 void module_timesteps(
00162     ctl_t * ctl,
00163     atm_t * atm,
00164     double *dt,
00165     double t);
00166
00168 void module_timesteps_init(
00169     ctl_t * ctl,
00170     atm_t * atm);
00171
00173 void module_wet_deposition(
00174     ctl_t * ctl,
00175     met_t * met0,
00176     met_t * met1,
00177     atm_t * atm,
00178     double *dt);
00179
00181 void write_output(
00182     const char *dirname,
00183     ctl_t * ctl,
00184     met_t * met0,
00185     met_t * met1,
00186     atm_t * atm,
00187     double t);
00188
00189 /* -----
00190     Main...
00191     ----- */
00192
00193 int main(
00194     int argc,
00195     char *argv[]) {
00196
00197     ctl_t ctl;
00198
00199     atm_t *atm;
00200
00201     cache_t *cache;
00202
00203     met_t *met0, *met1;
00204
00205     FILE *dirlist;
00206
00207     char dirname[LEN], filename[2 * LEN];

```

```

00208
00209     double *dt, *rs, t;
00210
00211     int num_devices = 0, ntask = -1, rank = 0, size = 1;
00212
00213     /* Start timers... */
00214     START_TIMERS;
00215
00216     /* Initialize MPI... */
00217 #ifdef MPI
00218     SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
00219     MPI_Init(&argc, &argv);
00220     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00221     MPI_Comm_size(MPI_COMM_WORLD, &size);
00222 #endif
00223
00224     /* Initialize GPUs... */
00225 #ifdef _OPENACC
00226     SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
00227     num_devices = acc_get_num_devices(acc_device_nvidia);
00228     if (num_devices <= 0)
00229         ERRMSG("Not running on a GPU device!");
00230     int device_num = rank % num_devices;
00231     acc_set_device_num(device_num, acc_device_nvidia);
00232     acc_device_t device_type = acc_get_device_type();
00233     acc_init(device_type);
00234 #endif
00235
00236     /* Check arguments... */
00237     if (argc < 4)
00238         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00239
00240     /* Open directory list... */
00241     if (!(dirlist = fopen(argv[1], "r")))
00242         ERRMSG("Cannot open directory list!");
00243
00244     /* Loop over directories... */
00245     while (fscanf(dirlist, "%4999s", dirname) != EOF) {
00246
00247         /* MPI parallelization... */
00248         if ((++ntask) % size != rank)
00249             continue;
00250
00251         /* -----
00252            Initialize model run...
00253            ----- */
00254
00255         /* Allocate... */
00256         SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
00257         ALLOC(atm, atm_t, 1);
00258         ALLOC(cache, cache_t, 1);
00259         ALLOC(met0, met_t, 1);
00260         ALLOC(met1, met_t, 1);
00261         ALLOC(dt, double,
00262              NP);
00263         ALLOC(rs, double,
00264              3 * NP + 1);
00265
00266         /* Create data region on GPUs... */
00267 #ifdef _OPENACC
00268         SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
00269         #pragma acc enter data create(atm[:1], cache[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3*NP])
00270 #endif
00271
00272         /* Read control parameters... */
00273         sprintf(filename, "%s/%s", dirname, argv[2]);
00274         read_ctl(filename, argc, argv, &ctl);
00275
00276         /* Read atmospheric data... */
00277         sprintf(filename, "%s/%s", dirname, argv[3]);
00278         if (!read_atm(filename, &ctl, atm))
00279             ERRMSG("Cannot open file!");
00280
00281         /* Initialize timesteps... */
00282         module_timesteps_init(&ctl, atm);
00283
00284         /* Update GPU... */
00285 #ifdef _OPENACC
00286         SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00287         #pragma acc update device(atm[:1], ctl)
00288 #endif
00289
00290         /* Initialize random number generator... */
00291         module_rng_init(ntask);
00292
00293         /* Initialize meteorological data... */
00294         get_met(&ctl, ctl.t_start, &met0, &met1);

```



```

00295     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00296         WARN("Violation of CFL criterion! Check DT_MOD!");
00297
00298     /* Initialize isosurface... */
00299     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00300         module_isosurf_init(&ctl, met0, met1, atm, cache);
00301
00302     /* Update GPU... */
00303 #ifdef _OPENACC
00304     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00305 #pragma acc update device(cache[:1])
00306 #endif
00307
00308     /* -----
00309      Loop over timesteps...
00310      ----- */
00311
00312     /* Loop over timesteps... */
00313     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00314          t += ctl.direction * ctl.dt_mod) {
00315
00316         /* Adjust length of final time step... */
00317         if (ctl.direction * (t - ctl.t_stop) > 0)
00318             t = ctl.t_stop;
00319
00320         /* Set time steps of air parcels... */
00321         module_timesteps(&ctl, atm, dt, t);
00322
00323         /* Get meteorological data... */
00324         if (t != ctl.t_start)
00325             get_met(&ctl, t, &met0, &met1);
00326
00327         /* Check initial positions... */
00328         module_position(&ctl, met0, met1, atm, dt);
00329
00330         /* Advection... */
00331         if (ctl.advect == 0)
00332             module_advect_mp(met0, met1, atm, dt);
00333         else
00334             module_advect_rk(met0, met1, atm, dt);
00335
00336         /* Turbulent diffusion... */
00337         if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00338             || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
00339             module_diffusion_turb(&ctl, atm, dt, rs);
00340
00341         /* Mesoscale diffusion... */
00342         if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
00343             module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00344
00345         /* Convection... */
00346         if (ctl.conv_cape >= 0
00347             && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
00348             module_convection(&ctl, met0, met1, atm, dt, rs);
00349
00350         /* Sedimentation... */
00351         if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00352             module_sedi(&ctl, met0, met1, atm, dt);
00353
00354         /* Isosurface... */
00355         if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00356             module_isosurf(&ctl, met0, met1, atm, cache);
00357
00358         /* Check final positions... */
00359         module_position(&ctl, met0, met1, atm, dt);
00360
00361         /* Interpolate meteorological data... */
00362         if (ctl.met_dt_out > 0
00363             && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.met_dt_out) == 0))
00364             module_meteo(&ctl, met0, met1, atm);
00365
00366         /* Decay of particle mass... */
00367         if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00368             module_decay(&ctl, atm, dt);
00369
00370         /* OH chemistry... */
00371         if (ctl.oh_chem_reaction != 0)
00372             module_oh_chem(&ctl, met0, met1, atm, dt);
00373
00374         /* Dry deposition... */
00375         if (ctl.dry_depo[0] > 0)
00376             module_dry_deposition(&ctl, met0, met1, atm, dt);
00377
00378         /* Wet deposition... */
00379         if ((ctl.wet_depo[0] > 0 || ctl.wet_depo[2] > 0)
00380             && (ctl.wet_depo[4] > 0 || ctl.wet_depo[6] > 0))
00381             module_wet_deposition(&ctl, met0, met1, atm, dt);

```

```

00382
00383     /* Boundary conditions... */
00384     if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
00385         module_bound_cond(&ctl, met0, met1, atm, dt);
00386
00387     /* Write output... */
00388     write_output(dirname, &ctl, met0, met1, atm, t);
00389 }
00390
00391 /* -----
00392 Finalize model run...
00393 ----- */
00394
00395 /* Report problem size... */
00396 LOG(1, "SIZE_NP = %d", atm->np);
00397 LOG(1, "SIZE_MPI_TASKS = %d", size);
00398 LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
00399 LOG(1, "SIZE_ACC_DEVICES = %d", num_devices);
00400
00401 /* Report memory usage... */
00402 LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
00403 LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
00404 LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
00405 LOG(1, "MEMORY_DYNAMIC = %g MByte", (sizeof(met_t)
00406                                     + 4 * NP * sizeof(double)
00407                                     + EX * EY * EP * sizeof(float)) /
00408     1024. / 1024.);
00409 LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * sizeof(double)
00410                                     + EX * EY * EP * sizeof(float)
00411                                     + 4 * GX * GY * GZ * sizeof(double)
00412                                     + 2 * GX * GY * GZ * sizeof(int)
00413                                     + 2 * GX * GY * sizeof(double)
00414                                     + GX * GY * sizeof(int)) / 1024. /
00415     1024.);
00416
00417 /* Delete data region on GPUs... */
00418 #ifdef _OPENACC
00419     SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
00420 #pragma acc exit data delete(ctl, atm, cache, met0, met1, dt, rs)
00421 #endif
00422
00423 /* Free... */
00424 SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
00425 free(atm);
00426 free(cache);
00427 free(met0);
00428 free(met1);
00429 free(dt);
00430 free(rs);
00431
00432 /* Report timers... */
00433 PRINT_TIMERS;
00434 }
00435
00436 /* Finalize MPI... */
00437 #ifdef MPI
00438     MPI_Finalize();
00439 #endif
00440
00441 /* Stop timers... */
00442 STOP_TIMERS;
00443
00444 return EXIT_SUCCESS;
00445 }
00446
00447 /*****
00448
00449 void module_advect_mp(
00450     met_t * met0,
00451     met_t * met1,
00452     atm_t * atm,
00453     double *dt) {
00454
00455     /* Set timer... */
00456     SELECT_TIMER("MODULE_ADVECTION", "PHYSICS", NVTX_GPU);
00457
00458     const int np = atm->np;
00459     #ifdef _OPENACC
00460     #pragma acc data present(met0, met1, atm, dt)
00461     #pragma acc parallel loop independent gang vector
00462     #else
00463     #pragma omp parallel for default(shared)
00464     #endif
00465     for (int ip = 0; ip < np; ip++)
00466         if (dt[ip] != 0) {
00467
00468             double u, v, w;

```

```

00469
00470 /* Interpolate meteorological data... */
00471 INTPOL_INIT;
00472 INTPOL_3D(u, 1);
00473 INTPOL_3D(v, 0);
00474 INTPOL_3D(w, 0);
00475
00476 /* Get position of the mid point... */
00477 double dtm = atm->time[ip] + 0.5 * dt[ip];
00478 double xm0 =
00479     atm->lon[ip] + DX2DEG(0.5 * dt[ip] * u / 1000., atm->lat[ip]);
00480 double xm1 = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v / 1000.);
00481 double xm2 = atm->p[ip] + 0.5 * dt[ip] * w;
00482
00483 /* Interpolate meteorological data for mid point... */
00484 intpol_met_time_3d(met0, met0->u, met1, met1->u,
00485     dtm, xm2, xm0, xm1, &u, ci, cw, 1);
00486 intpol_met_time_3d(met0, met0->v, met1, met1->v,
00487     dtm, xm2, xm0, xm1, &v, ci, cw, 0);
00488 intpol_met_time_3d(met0, met0->w, met1, met1->w,
00489     dtm, xm2, xm0, xm1, &w, ci, cw, 0);
00490
00491 /* Save new position... */
00492 atm->time[ip] += dt[ip];
00493 atm->lon[ip] += DX2DEG(dt[ip] * u / 1000., xm1);
00494 atm->lat[ip] += DY2DEG(dt[ip] * v / 1000.);
00495 atm->p[ip] += dt[ip] * w;
00496 }
00497 }
00498
00499 /*****
00500
00501 void module_advect_rk(
00502     met_t * met0,
00503     met_t * met1,
00504     atm_t * atm,
00505     double *dt) {
00506
00507     /* Set timer... */
00508     SELECT_TIMER("MODULE_ADEVECTION", "PHYSICS", NVTX_GPU);
00509
00510     const int np = atm->np;
00511 #ifdef _OPENACC
00512 #pragma acc data present(met0,met1,atm,dt)
00513 #pragma acc parallel loop independent gang vector
00514 #else
00515 #pragma omp parallel for default(shared)
00516 #endif
00517     for (int ip = 0; ip < np; ip++)
00518         if (dt[ip] != 0) {
00519
00520             /* Init... */
00521             double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];
00522             INTPOL_INIT;
00523
00524             /* Loop over integration nodes... */
00525             for (int i = 0; i < 4; i++) {
00526
00527                 /* Set position... */
00528                 if (i == 0) {
00529                     dts = 0.0;
00530                     x[0] = atm->lon[ip];
00531                     x[1] = atm->lat[ip];
00532                     x[2] = atm->p[ip];
00533                 } else {
00534                     dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
00535                     x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
00536                     x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
00537                     x[2] = atm->p[ip] + dts * w[i - 1];
00538                 }
00539
00540                 /* Interpolate meteo data... */
00541                 double tm = atm->time[ip] + dts;
00542                 intpol_met_time_3d(met0, met0->u, met1, met1->u, tm, x[2], x[0], x[1],
00543                     &u[i], ci, cw, 1);
00544                 intpol_met_time_3d(met0, met0->v, met1, met1->v, tm, x[2], x[0], x[1],
00545                     &v[i], ci, cw, 0);
00546                 intpol_met_time_3d(met0, met0->w, met1, met1->w, tm, x[2], x[0], x[1],
00547                     &w[i], ci, cw, 0);
00548
00549                 /* Get mean wind... */
00550                 double k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
00551                 um += k * u[i];
00552                 vm += k * v[i];
00553                 wm += k * w[i];
00554             }
00555

```

```

00556      /* Set new position... */
00557      atm->time[ip] += dt[ip];
00558      atm->lon[ip] += DX2DEG(dt[ip] * um / 1000., atm->lat[ip]);
00559      atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
00560      atm->p[ip] += dt[ip] * wm;
00561  }
00562 }
00563
00564 /*****
00565 void module_bound_cond(
00566     ctl_t * ctl,
00567     met_t * met0,
00568     met_t * met1,
00569     atm_t * atm,
00570     double *dt) {
00571     /* Set timer... */
00572     SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);
00573
00574     /* Check quantity flags... */
00575     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00576         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00577
00578     const int np = atm->np;
00579 #ifdef _OPENACC
00580 #pragma acc data present(ctl,met0,met1,atm,dt)
00581 #pragma acc parallel loop independent gang vector
00582 #else
00583 #pragma omp parallel for default(shared)
00584 #endif
00585     for (int ip = 0; ip < np; ip++)
00586         if (dt[ip] != 0) {
00587             double ps;
00588
00589             /* Check latitude and pressure range... */
00590             if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
00591                 || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
00592                 continue;
00593
00594             /* Check surface layer... */
00595             if (ctl->bound_dps > 0) {
00596                 /* Get surface pressure... */
00597                 INTPOL_INIT;
00598                 INTPOL_2D(ps, 1);
00599
00600                 /* Check whether particle is above the surface layer... */
00601                 if (atm->p[ip] < ps - ctl->bound_dps)
00602                     continue;
00603             }
00604
00605             /* Set mass and volume mixing ratio... */
00606             if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
00607                 atm->q[ctl->qnt_m][ip] = ctl->bound_mass;
00608             if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
00609                 atm->q[ctl->qnt_vmr][ip] = ctl->bound_vmr;
00610         }
00611 }
00612
00613 /*****
00614 void module_convection(
00615     ctl_t * ctl,
00616     met_t * met0,
00617     met_t * met1,
00618     atm_t * atm,
00619     double *dt,
00620     double *rs) {
00621     /* Set timer... */
00622     SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);
00623
00624     /* Create random numbers... */
00625     module_rng(rs, (size_t) atm->np, 0);
00626
00627     const int np = atm->np;
00628 #ifdef _OPENACC
00629 #pragma acc data present(ctl,met0,met1,atm,dt,rs)
00630 #pragma acc parallel loop independent gang vector
00631 #else
00632 #pragma omp parallel for default(shared)
00633 #endif
00634     for (int ip = 0; ip < np; ip++)
00635         if (dt[ip] != 0) {
00636
00637
00638
00639
00640
00641
00642

```

```

00643     double cape, cin, pel, ps;
00644
00645     /* Interpolate CAPE... */
00646     INTPOL_INIT;
00647     INTPOL_2D(cape, 1);
00648
00649     /* Check threshold... */
00650     if (isfinite(cape) && cape >= ctl->conv_cape) {
00651
00652         /* Check CIN... */
00653         if (ctl->conv_cin > 0) {
00654             INTPOL_2D(cin, 0);
00655             if (isfinite(cin) && cin >= ctl->conv_cin)
00656                 continue;
00657         }
00658
00659         /* Interpolate equilibrium level... */
00660         INTPOL_2D(pel, 0);
00661
00662         /* Check whether particle is above cloud top... */
00663         if (!isfinite(pel) || atm->p[ip] < pel)
00664             continue;
00665
00666         /* Set pressure range for mixing... */
00667         double pbot = atm->p[ip];
00668         double ptop = atm->p[ip];
00669         if (ctl->conv_mix_bot == 1) {
00670             INTPOL_2D(ps, 0);
00671             pbot = ps;
00672         }
00673         if (ctl->conv_mix_top == 1)
00674             ptop = pel;
00675
00676         /* Limit vertical velocity... */
00677         if (ctl->conv_wmax > 0 || ctl->conv_wcape) {
00678             double z = Z(atm->p[ip]);
00679             double wmax = (ctl->conv_wcape) ? sqrt(2. * cape) : ctl->conv_wmax;
00680             double pmax = P(z - wmax * dt[ip] / 1000.);
00681             double pmin = P(z + wmax * dt[ip] / 1000.);
00682             ptop = GSL_MAX(ptop, pmin);
00683             pbot = GSL_MIN(pbot, pmax);
00684         }
00685
00686         /* Vertical mixing... */
00687         atm->p[ip] = pbot + (ptop - pbot) * rs[ip];
00688     }
00689 }
00690 }
00691
00692 /*****
00693
00694 void module_decay(
00695     ctl_t * ctl,
00696     atm_t * atm,
00697     double *dt) {
00698
00699     /* Set timer... */
00700     SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);
00701
00702     /* Check quantity flags... */
00703     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00704         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00705
00706     const int np = atm->np;
00707     #ifdef _OPENACC
00708     #pragma acc data present(ctl,atm,dt)
00709     #pragma acc parallel loop independent gang vector
00710     #else
00711     #pragma omp parallel for default(shared)
00712     #endif
00713     for (int ip = 0; ip < np; ip++)
00714         if (dt[ip] != 0) {
00715
00716             /* Get weighting factor... */
00717             double w = tropo_weight(atm->time[ip], atm->lat[ip], atm->p[ip]);
00718
00719             /* Set lifetime... */
00720             double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00721
00722             /* Calculate exponential decay... */
00723             double aux = exp(-dt[ip] / tdec);
00724             if (ctl->qnt_m >= 0)
00725                 atm->q[ctl->qnt_m][ip] *= aux;
00726             if (ctl->qnt_vmr >= 0)
00727                 atm->q[ctl->qnt_vmr][ip] *= aux;
00728         }
00729 }

```

```

00730
00731 /*****
00732
00733 void module_diffusion_meso(
00734     ctl_t * ctl,
00735     met_t * met0,
00736     met_t * met1,
00737     atm_t * atm,
00738     cache_t * cache,
00739     double *dt,
00740     double *rs) {
00741
00742     /* Set timer... */
00743     SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);
00744
00745     /* Create random numbers... */
00746     module_rng(rs, 3 * (size_t) atm->np, 1);
00747
00748     const int np = atm->np;
00749     #ifdef _OPENACC
00750     #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00751     #pragma acc parallel loop independent gang vector
00752     #else
00753     #pragma omp parallel for default(shared)
00754     #endif
00755     for (int ip = 0; ip < np; ip++)
00756         if (dt[ip] != 0) {
00757
00758             /* Get indices... */
00759             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00760             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00761             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00762
00763             /* Caching of wind standard deviations... */
00764             if (cache->tsig[ix][iy][iz] != met0->time) {
00765
00766                 /* Collect local wind data... */
00767                 int n = 0;
00768                 float u[16], v[16], w[16];
00769                 for (int i = 0; i < 2; i++)
00770                     for (int j = 0; j < 2; j++)
00771                         for (int k = 0; k < 2; k++) {
00772                             u[n] = met0->u[ix + i][iy + j][iz + k];
00773                             v[n] = met0->v[ix + i][iy + j][iz + k];
00774                             w[n] = met0->w[ix + i][iy + j][iz + k];
00775                             n++;
00776                             u[n] = met1->u[ix + i][iy + j][iz + k];
00777                             v[n] = met1->v[ix + i][iy + j][iz + k];
00778                             w[n] = met1->w[ix + i][iy + j][iz + k];
00779                             n++;
00780                         }
00781
00782                 /* Get standard deviations of local wind data... */
00783                 cache->uvwsig[ix][iy][iz][0] = stddev(u, n);
00784                 cache->uvwsig[ix][iy][iz][1] = stddev(v, n);
00785                 cache->uvwsig[ix][iy][iz][2] = stddev(w, n);
00786
00787                 /* Save new time... */
00788                 cache->tsig[ix][iy][iz] = met0->time;
00789             }
00790
00791             /* Set temporal correlations for mesoscale fluctuations... */
00792             double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00793             double r2 = sqrt(1 - r * r);
00794
00795             /* Calculate horizontal mesoscale wind fluctuations... */
00796             if (ctl->turb_mesox > 0) {
00797                 cache->uvwp[ip][0] = (float)
00798                     (r * cache->uvwp[ip][0]
00799                      +
00800                     r2 * rs[3 * ip] * ctl->turb_mesox * cache->uvwsig[ix][iy][iz][0]);
00801                 atm->lon[ip] +=
00802                     DX2DEG(cache->uvwp[ip][0] * dt[ip] / 1000., atm->lat[ip]);
00803
00804                 cache->uvwp[ip][1] = (float)
00805                     (r * cache->uvwp[ip][1]
00806                      + r2 * rs[3 * ip +
00807                      1] * ctl->turb_mesox * cache->uvwsig[ix][iy][iz][1]);
00808                 atm->lat[ip] += DY2DEG(cache->uvwp[ip][1] * dt[ip] / 1000.);
00809             }
00810
00811             /* Calculate vertical mesoscale wind fluctuations... */
00812             if (ctl->turb_mesoz > 0) {
00813                 cache->uvwp[ip][2] = (float)
00814                     (r * cache->uvwp[ip][2]
00815                      + r2 * rs[3 * ip +
00816                      2] * ctl->turb_mesoz * cache->uvwsig[ix][iy][iz][2]);

```

```

00817         atm->p[ip] += cache->uvwp[ip][2] * dt[ip];
00818     }
00819 }
00820 }
00821
00822 /*****
00823
00824 void module_diffusion_turb(
00825     ctl_t * ctl,
00826     atm_t * atm,
00827     double *dt,
00828     double *rs) {
00829
00830     /* Set timer... */
00831     SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);
00832
00833     /* Create random numbers... */
00834     module_rng(rs, 3 * (size_t) atm->np, 1);
00835
00836     const int np = atm->np;
00837     #ifdef _OPENACC
00838     #pragma acc data present(ctl,atm,dt,rs)
00839     #pragma acc parallel loop independent gang vector
00840     #else
00841     #pragma omp parallel for default(shared)
00842     #endif
00843     for (int ip = 0; ip < np; ip++)
00844         if (dt[ip] != 0) {
00845
00846             /* Get weighting factor... */
00847             double w = tropo_weight(atm->time[ip], atm->lat[ip], atm->p[ip]);
00848
00849             /* Set diffusivity... */
00850             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00851             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00852
00853             /* Horizontal turbulent diffusion... */
00854             if (dx > 0) {
00855                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00856                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00857                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00858             }
00859
00860             /* Vertical turbulent diffusion... */
00861             if (dz > 0) {
00862                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00863                 atm->p[ip]
00864                     += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00865             }
00866         }
00867 }
00868
00869 /*****
00870
00871 void module_dry_deposition(
00872     ctl_t * ctl,
00873     met_t * met0,
00874     met_t * met1,
00875     atm_t * atm,
00876     double *dt) {
00877
00878     /* Set timer... */
00879     SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);
00880
00881     /* Width of the surface layer [hPa]. */
00882     const double dp = 30.;
00883
00884     /* Check quantity flags... */
00885     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00886         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00887
00888     const int np = atm->np;
00889     #ifdef _OPENACC
00890     #pragma acc data present(ctl,met0,met1,atm,dt)
00891     #pragma acc parallel loop independent gang vector
00892     #else
00893     #pragma omp parallel for default(shared)
00894     #endif
00895     for (int ip = 0; ip < np; ip++)
00896         if (dt[ip] != 0) {
00897
00898             double ps, t, v_dep;
00899
00900             /* Get surface pressure... */
00901             INTPOL_INIT;
00902             INTPOL_2D(ps, 1);
00903

```

```

00904      /* Check whether particle is above the surface layer... */
00905      if (atm->p[ip] < ps - dp)
00906          continue;
00907
00908      /* Set width of surface layer... */
00909      double dz = 1000. * (Z(ps - dp) - Z(ps));
00910
00911      /* Calculate sedimentation velocity for particles... */
00912      if (ctl->qnt_r > 0 && ctl->qnt_rho > 0) {
00913
00914          /* Get temperature... */
00915          INTPOL_3D(t, 1);
00916
00917          /* Set deposition velocity... */
00918          v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
00919                      atm->q[ctl->qnt_rho][ip]);
00920      }
00921
00922      /* Use explicit sedimentation velocity for gases... */
00923      else
00924          v_dep = ctl->dry_depo[0];
00925
00926      /* Calculate loss of mass based on deposition velocity... */
00927      double aux = exp(-dt[ip] * v_dep / dz);
00928      if (ctl->qnt_m >= 0)
00929          atm->q[ctl->qnt_m][ip] *= aux;
00930      if (ctl->qnt_vmr >= 0)
00931          atm->q[ctl->qnt_vmr][ip] *= aux;
00932      }
00933 }
00934
00935 /*****
00936
00937 void module_isosurf_init(
00938     ctl_t * ctl,
00939     met_t * met0,
00940     met_t * met1,
00941     atm_t * atm,
00942     cache_t * cache) {
00943
00944     FILE *in;
00945
00946     char line[LEN];
00947
00948     double t;
00949
00950     /* Set timer... */
00951     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
00952
00953     /* Init... */
00954     INTPOL_INIT;
00955
00956     /* Save pressure... */
00957     if (ctl->isosurf == 1)
00958         for (int ip = 0; ip < atm->np; ip++)
00959             cache->iso_var[ip] = atm->p[ip];
00960
00961     /* Save density... */
00962     else if (ctl->isosurf == 2)
00963         for (int ip = 0; ip < atm->np; ip++) {
00964             INTPOL_3D(t, 1);
00965             cache->iso_var[ip] = atm->p[ip] / t;
00966         }
00967
00968     /* Save potential temperature... */
00969     else if (ctl->isosurf == 3)
00970         for (int ip = 0; ip < atm->np; ip++) {
00971             INTPOL_3D(t, 1);
00972             cache->iso_var[ip] = THETA(atm->p[ip], t);
00973         }
00974
00975     /* Read balloon pressure data... */
00976     else if (ctl->isosurf == 4) {
00977
00978         /* Write info... */
00979         LOG(1, "Read balloon pressure data: %s", ctl->balloon);
00980
00981         /* Open file... */
00982         if (!(in = fopen(ctl->balloon, "r")))
00983             ERRMSG("Cannot open file!");
00984
00985         /* Read pressure time series... */
00986         while (fgets(line, LEN, in))
00987             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00988                       &(cache->iso_ps[cache->iso_n])) == 2)
00989                 if (++cache->iso_n > NP)
00990                     ERRMSG("Too many data points!");

```



```

00991
00992     /* Check number of points... */
00993     if (cache->iso_n < 1)
00994         ERRMSG("Could not read any data!");
00995
00996     /* Close file... */
00997     fclose(in);
00998 }
00999 }
01000
01001 /*****
01002
01003 void module_isosurf(
01004     ctl_t * ctl,
01005     met_t * met0,
01006     met_t * met1,
01007     atm_t * atm,
01008     cache_t * cache) {
01009
01010     /* Set timer... */
01011     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01012
01013     const int np = atm->np;
01014     #ifdef _OPENACC
01015     #pragma acc data present(ctl,met0,met1,atm,cache)
01016     #pragma acc parallel loop independent gang vector
01017     #else
01018     #pragma omp parallel for default(shared)
01019     #endif
01020     for (int ip = 0; ip < np; ip++) {
01021
01022         double t;
01023
01024         /* Init... */
01025         INTPOL_INIT;
01026
01027         /* Restore pressure... */
01028         if (ctl->isosurf == 1)
01029             atm->p[ip] = cache->iso_var[ip];
01030
01031         /* Restore density... */
01032         else if (ctl->isosurf == 2) {
01033             INTPOL_3D(t, 1);
01034             atm->p[ip] = cache->iso_var[ip] * t;
01035         }
01036
01037         /* Restore potential temperature... */
01038         else if (ctl->isosurf == 3) {
01039             INTPOL_3D(t, 1);
01040             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
01041         }
01042
01043         /* Interpolate pressure... */
01044         else if (ctl->isosurf == 4) {
01045             if (atm->time[ip] <= cache->iso_ts[0])
01046                 atm->p[ip] = cache->iso_ps[0];
01047             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
01048                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
01049             else {
01050                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
01051                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
01052                             cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
01053                             atm->time[ip]);
01054             }
01055         }
01056     }
01057 }
01058
01059 /*****
01060
01061 void module_meteo(
01062     ctl_t * ctl,
01063     met_t * met0,
01064     met_t * met1,
01065     atm_t * atm) {
01066
01067     /* Set timer... */
01068     SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);
01069
01070     /* Check quantity flags... */
01071     if (ctl->qnt_tsts >= 0)
01072         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01073             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01074
01075     const int np = atm->np;
01076     #ifdef _OPENACC
01077     #pragma acc data present(ctl,met0,met1,atm)

```

```

01078 #pragma acc parallel loop independent gang vector
01079 #else
01080 #pragma omp parallel for default(shared)
01081 #endif
01082     for (int ip = 0; ip < np; ip++) {
01083
01084         double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
01085             cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
01086
01087         /* Interpolate meteorological data... */
01088         INTPOL_INIT;
01089         INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
01090
01091         /* Set quantities... */
01092         SET_ATM(qnt_ps, ps);
01093         SET_ATM(qnt_ts, ts);
01094         SET_ATM(qnt_zs, zs);
01095         SET_ATM(qnt_us, us);
01096         SET_ATM(qnt_vs, vs);
01097         SET_ATM(qnt_pbl, pbl);
01098         SET_ATM(qnt_pt, pt);
01099         SET_ATM(qnt_tt, tt);
01100         SET_ATM(qnt_zt, zt);
01101         SET_ATM(qnt_h2ot, h2ot);
01102         SET_ATM(qnt_p, atm->p[ip]);
01103         SET_ATM(qnt_z, z);
01104         SET_ATM(qnt_t, t);
01105         SET_ATM(qnt_u, u);
01106         SET_ATM(qnt_v, v);
01107         SET_ATM(qnt_w, w);
01108         SET_ATM(qnt_h2o, h2o);
01109         SET_ATM(qnt_o3, o3);
01110         SET_ATM(qnt_lwc, lwc);
01111         SET_ATM(qnt_iwc, iwc);
01112         SET_ATM(qnt_pct, pct);
01113         SET_ATM(qnt_pcb, pcb);
01114         SET_ATM(qnt_cl, cl);
01115         SET_ATM(qnt_plcl, plcl);
01116         SET_ATM(qnt_plfc, plfc);
01117         SET_ATM(qnt_pel, pel);
01118         SET_ATM(qnt_cape, cape);
01119         SET_ATM(qnt_cin, cin);
01120         SET_ATM(qnt_hno3, clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]));
01121         SET_ATM(qnt_oh, clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01122         SET_ATM(qnt_vh, sqrt(u * u + v * v));
01123         SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01124         SET_ATM(qnt_psat, PSAT(t));
01125         SET_ATM(qnt_psice, PSICE(t));
01126         SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
01127         SET_ATM(qnt_sh, SH(h2o));
01128         SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
01129         SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01130         SET_ATM(qnt_theta, THETA(atm->p[ip], t));
01131         SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
01132         SET_ATM(qnt_tvirt, TVIRT(t, h2o));
01133         SET_ATM(qnt_lapse, lapse_rate(t, h2o));
01134         SET_ATM(qnt_pv, pv);
01135         SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
01136         SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
01137         SET_ATM(qnt_tnat,
01138             nat_temperature(atm->p[ip], h2o,
01139                 clim_hno3(atm->time[ip], atm->lat[ip],
01140                     atm->p[ip])));
01141         SET_ATM(qnt_tsts,
01142             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01143     }
01144 }
01145
01146 /*****
01147
01148 void module_oh_chem(
01149     ctl_t * ctl,
01150     met_t * met0,
01151     met_t * met1,
01152     atm_t * atm,
01153     double *dt) {
01154
01155     /* Set timer... */
01156     SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);
01157
01158     /* Check quantity flags... */
01159     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01160         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01161
01162     const int np = atm->np;
01163     #ifdef _OPENACC
01164     #pragma acc data present(ctl,met0,met1,atm,dt)

```

```

01165 #pragma acc parallel loop independent gang vector
01166 #else
01167 #pragma omp parallel for default(shared)
01168 #endif
01169     for (int ip = 0; ip < np; ip++)
01170         if (dt[ip] != 0) {
01171             /* Get temperature... */
01172             double t;
01173             INTPOL_INIT;
01174             INTPOL_3D(t, 1);
01175
01176             /* Use constant reaction rate... */
01177             double k = GSL_NAN;
01178             if (ctl->oh_chem_reaction == 1)
01179                 k = ctl->oh_chem[0];
01180
01181             /* Calculate bimolecular reaction rate... */
01182             else if (ctl->oh_chem_reaction == 2)
01183                 k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);
01184
01185             /* Calculate termolecular reaction rate... */
01186             if (ctl->oh_chem_reaction == 3) {
01187                 /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
01188                 double M = 7.243e21 * (atm->p[ip] / 1000.) / t;
01189
01190                 /* Calculate rate coefficient for X + OH + M -> XOH + M
01191                  (JPL Publication 19-05) ... */
01192                 double k0 = ctl->oh_chem[0] *
01193                     (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
01194                 double ki = ctl->oh_chem[2] *
01195                     (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
01196                 double c = log10(k0 * M / ki);
01197                 k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01198             }
01199
01200             /* Calculate exponential decay... */
01201             double aux
01202                 = exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]));
01203             if (ctl->qnt_m >= 0)
01204                 atm->q[ctl->qnt_m][ip] *= aux;
01205             if (ctl->qnt_vmr >= 0)
01206                 atm->q[ctl->qnt_vmr][ip] *= aux;
01207         }
01208     }
01209 }
01210
01211
01212 /*****
01213
01214 void module_position(
01215     ctl_t * ctl,
01216     met_t * met0,
01217     met_t * met1,
01218     atm_t * atm,
01219     double *dt) {
01220
01221     /* Set timer... */
01222     SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);
01223
01224     const int np = atm->np;
01225     #ifdef _OPENACC
01226     #pragma acc data present (met0, met1, atm, dt)
01227     #pragma acc parallel loop independent gang vector
01228     #else
01229     #pragma omp parallel for default(shared)
01230     #endif
01231     for (int ip = 0; ip < np; ip++)
01232         if (dt[ip] != 0) {
01233             /* Init... */
01234             double ps;
01235             INTPOL_INIT;
01236
01237             /* Calculate modulo... */
01238             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01239             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01240
01241             /* Check latitude... */
01242             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01243                 if (atm->lat[ip] > 90) {
01244                     atm->lat[ip] = 180 - atm->lat[ip];
01245                     atm->lon[ip] += 180;
01246                 }
01247                 if (atm->lat[ip] < -90) {
01248                     atm->lat[ip] = -180 - atm->lat[ip];
01249                     atm->lon[ip] += 180;
01250                 }
01251             }

```

```

01252     }
01253
01254     /* Check longitude... */
01255     while (atm->lon[ip] < -180)
01256         atm->lon[ip] += 360;
01257     while (atm->lon[ip] >= 180)
01258         atm->lon[ip] -= 360;
01259
01260     /* Check pressure... */
01261     if (atm->p[ip] < met0->p[met0->np - 1]) {
01262         if (ctl->reflect)
01263             atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
01264         else
01265             atm->p[ip] = met0->p[met0->np - 1];
01266     } else if (atm->p[ip] > 300.) {
01267         INTPOL_2D(ps, 1);
01268         if (atm->p[ip] > ps) {
01269             if (ctl->reflect)
01270                 atm->p[ip] = 2. * ps - atm->p[ip];
01271             else
01272                 atm->p[ip] = ps;
01273         }
01274     }
01275 }
01276 }
01277
01278 /*****
01279
01280 void module_rng_init(
01281     int ntask) {
01282
01283     /* Initialize random number generator... */
01284     #ifdef _OPENACC
01285
01286         if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
01287             != CURAND_STATUS_SUCCESS)
01288             ERRMSG("Cannot create random number generator!");
01289         if (curandSetPseudoRandomGeneratorSeed(rng, ntask)
01290             != CURAND_STATUS_SUCCESS)
01291             ERRMSG("Cannot set seed for random number generator!");
01292         if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01293             != CURAND_STATUS_SUCCESS)
01294             ERRMSG("Cannot set stream for random number generator!");
01295     #else
01296
01297         gsl_rng_env_setup();
01298         if (omp_get_max_threads() > NTHREADS)
01299             ERRMSG("Too many threads!");
01300         for (int i = 0; i < NTHREADS; i++) {
01301             rng[i] = gsl_rng_alloc(gsl_rng_default);
01302             gsl_rng_set(rng[i], gsl_rng_default_seed
01303                 + (long unsigned) (ntask * NTHREADS + i));
01304         }
01305     #endif
01306 }
01307 #endif
01308 }
01309
01310 /*****
01311
01312 void module_rng(
01313     double *rs,
01314     size_t n,
01315     int method) {
01316
01317     #ifdef _OPENACC
01318
01319     #pragma acc host_data use_device(rs)
01320     {
01321         /* Uniform distribution... */
01322         if (method == 0) {
01323             if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n))
01324                 != CURAND_STATUS_SUCCESS)
01325                 ERRMSG("Cannot create random numbers!");
01326         }
01327
01328         /* Normal distribution... */
01329         else if (method == 1) {
01330             if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0)
01331                 != CURAND_STATUS_SUCCESS)
01332                 ERRMSG("Cannot create random numbers!");
01333         }
01334     }
01335 #else
01336
01337     /* Uniform distribution... */

```

```

01339     if (method == 0) {
01340 #pragma omp parallel for default(shared)
01341         for (size_t i = 0; i < n; ++i)
01342             rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01343     }
01344
01345     /* Normal distribution... */
01346     else if (method == 1) {
01347 #pragma omp parallel for default(shared)
01348         for (size_t i = 0; i < n; ++i)
01349             rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01350     }
01351 #endif
01352 }
01353
01354
01355 /*****
01356
01357 void module_sedi(
01358     ctl_t * ctl,
01359     met_t * met0,
01360     met_t * met1,
01361     atm_t * atm,
01362     double *dt) {
01363
01364     /* Set timer... */
01365     SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);
01366
01367     const int np = atm->np;
01368 #ifdef _OPENACC
01369 #pragma acc data present(ctl,met0,met1,atm,dt)
01370 #pragma acc parallel loop independent gang vector
01371 #else
01372 #pragma omp parallel for default(shared)
01373 #endif
01374     for (int ip = 0; ip < np; ip++)
01375         if (dt[ip] != 0) {
01376
01377             /* Get temperature... */
01378             double t;
01379             INTPOL_INIT;
01380             INTPOL_3D(t, 1);
01381
01382             /* Sedimentation velocity... */
01383             double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_r][ip],
01384                             atm->q[ctl->qnt_rho][ip]);
01385
01386             /* Calculate pressure change... */
01387             atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01388         }
01389 }
01390
01391 /*****
01392
01393 void module_timesteps(
01394     ctl_t * ctl,
01395     atm_t * atm,
01396     double *dt,
01397     double t) {
01398
01399     /* Set timer... */
01400     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01401
01402     const int np = atm->np;
01403 #ifdef _OPENACC
01404 #pragma acc data present(ctl,atm,dt)
01405 #pragma acc parallel loop independent gang vector
01406 #else
01407 #pragma omp parallel for default(shared)
01408 #endif
01409     for (int ip = 0; ip < np; ip++) {
01410         if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
01411             && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
01412             && ctl->direction * (atm->time[ip] - t) < 0))
01413             dt[ip] = t - atm->time[ip];
01414         else
01415             dt[ip] = 0.0;
01416     }
01417 }
01418
01419 /*****
01420
01421 void module_timesteps_init(
01422     ctl_t * ctl,
01423     atm_t * atm) {
01424
01425     /* Set timer... */

```

```

01426     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01427
01428     /* Set start time... */
01429     if (ctl->direction == 1) {
01430         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01431         if (ctl->t_stop > 1e99)
01432             ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01433     } else {
01434         ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01435         if (ctl->t_stop > 1e99)
01436             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01437     }
01438
01439     /* Check time interval... */
01440     if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
01441         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
01442
01443     /* Round start time... */
01444     if (ctl->direction == 1)
01445         ctl->t_start = floor(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01446     else
01447         ctl->t_start = ceil(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01448 }
01449
01450 /*****
01451 void module_wet_deposition(
01452     ctl_t * ctl,
01453     met_t * met0,
01454     met_t * met1,
01455     atm_t * atm,
01456     double *dt) {
01457
01458     /* Set timer... */
01459     SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);
01460
01461     /* Check quantity flags... */
01462     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01463         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01464
01465     const int np = atm->np;
01466     #ifdef _OPENACC
01467     #pragma acc data present(ctl,met0,met1,atm,dt)
01468     #pragma acc parallel loop independent gang vector
01469     #else
01470     #pragma omp parallel for default(shared)
01471     #endif
01472     for (int ip = 0; ip < np; ip++)
01473         if (dt[ip] != 0) {
01474             double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;
01475
01476             int inside;
01477
01478             /* Check whether particle is below cloud top... */
01479             INTPOL_INIT;
01480             INTPOL_2D(pct, 1);
01481             if (!isfinite(pct) || atm->p[ip] <= pct)
01482                 continue;
01483
01484             /* Get cloud bottom pressure... */
01485             INTPOL_2D(pcb, 0);
01486
01487             /* Estimate precipitation rate (Pisso et al., 2019)... */
01488             INTPOL_2D(cl, 0);
01489             double Is = pow(2. * cl, 1. / 0.36);
01490             if (Is < 0.01)
01491                 continue;
01492
01493             /* Check whether particle is inside or below cloud... */
01494             INTPOL_3D(lwc, 1);
01495             INTPOL_3D(iwc, 0);
01496             inside = (iwc > 0 || lwc > 0);
01497
01498             /* Calculate in-cloud scavenging coefficient... */
01499             if (inside) {
01500                 /* Use exponential dependency for particles... */
01501                 if (ctl->wet_depo[0] > 0)
01502                     lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01503                 /* Use Henry's law for gases... */
01504                 else if (ctl->wet_depo[2] > 0) {
01505                     /* Get temperature... */
01506                     INTPOL_3D(t, 0);

```

```

01513      /* Get Henry's constant (Sander, 2015)... */
01514      h = ctl->wet_depo[2]
01515          * exp(ctl->wet_depo[3] * (1. / t - 1. / 298.15));
01516
01517      /* Estimate depth of cloud layer... */
01518      dz = 1e3 * (Z(pct) - Z(pcb));
01519
01520      /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01521      lambda = h * RI * t * Is / 3.6e6 / dz;
01522  }
01523  }
01524
01525  /* Calculate below-cloud scavenging coefficient... */
01526  else {
01527
01528      /* Use exponential dependency for particles... */
01529      if (ctl->wet_depo[4] > 0)
01530          lambda = ctl->wet_depo[4] * pow(Is, ctl->wet_depo[5]);
01531
01532      /* Use Henry's law for gases... */
01533      else if (ctl->wet_depo[6] > 0) {
01534
01535          /* Get temperature... */
01536          INTPOL_3D(t, 0);
01537
01538          /* Get Henry's constant (Sander, 2015)... */
01539          h = ctl->wet_depo[6]
01540              * exp(ctl->wet_depo[7] * (1. / t - 1. / 298.15));
01541
01542          /* Estimate depth of cloud layer... */
01543          dz = 1e3 * (Z(pct) - Z(pcb));
01544
01545          /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
01546          lambda = h * RI * t * Is / 3.6e6 / dz;
01547      }
01548  }
01549
01550  /* Calculate exponential decay of mass... */
01551  double aux = exp(-dt[ip] * lambda);
01552  if (ctl->qnt_m >= 0)
01553      atm->q[ctl->qnt_m][ip] *= aux;
01554  if (ctl->qnt_vmr >= 0)
01555      atm->q[ctl->qnt_vmr][ip] *= aux;
01556  }
01557 }
01558
01559 /*****
01560
01561 void write_output(
01562     const char *dirname,
01563     ctl_t * ctl,
01564     met_t * met0,
01565     met_t * met1,
01566     atm_t * atm,
01567     double t) {
01568
01569     char filename[2 * LEN];
01570
01571     double r;
01572
01573     int year, mon, day, hour, min, sec;
01574
01575     /* Get time... */
01576     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01577
01578     /* Update host... */
01579 #ifdef _OPENACC
01580     if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
01581         || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
01582         || (ctl->csi_basename[0] != '-' || ctl->ens_basename[0] != '-'
01583             || ctl->prof_basename[0] != '-' || ctl->sample_basename[0] != '-'
01584             || ctl->stat_basename[0] != '-')) {
01585         SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
01586         #pragma acc update host(atm[:1])
01587     }
01588 #endif
01589
01590     /* Write atmospheric data... */
01591     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01592         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01593             dirname, ctl->atm_basename, year, mon, day, hour, min);
01594         write_atm(filename, ctl, atm, t);
01595     }
01596
01597     /* Write gridded data... */
01598     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01599         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",

```

```

01600         dirname, ctl->grid_basename, year, mon, day, hour, min);
01601     write_grid(filename, ctl, met0, met1, atm, t);
01602 }
01603
01604 /* Write CSI data... */
01605 if (ctl->csi_basename[0] != '-') {
01606     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01607     write_csi(filename, ctl, atm, t);
01608 }
01609
01610 /* Write ensemble data... */
01611 if (ctl->ens_basename[0] != '-') {
01612     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01613     write_ens(filename, ctl, atm, t);
01614 }
01615
01616 /* Write profile data... */
01617 if (ctl->prof_basename[0] != '-') {
01618     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01619     write_prof(filename, ctl, met0, met1, atm, t);
01620 }
01621
01622 /* Write sample data... */
01623 if (ctl->sample_basename[0] != '-') {
01624     sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
01625     write_sample(filename, ctl, met0, met1, atm, t);
01626 }
01627
01628 /* Write station data... */
01629 if (ctl->stat_basename[0] != '-') {
01630     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01631     write_station(filename, ctl, atm, t);
01632 }
01633 }

```

5.45 tropo.c File Reference

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

Functions

- void [add_text_attribute](#) (int ncid, char *varname, char *attrname, char *text)
- int [main](#) (int argc, char *argv[])

5.45.1 Detailed Description

Create tropopause data set from meteorological data.

Definition in file [tropo.c](#).

5.45.2 Function Documentation

5.45.2.1 add_text_attribute() void add_text_attribute (

```

    int ncid,
    char * varname,
    char * attrname,
    char * text )

```

Definition at line 337 of file [tropo.c](#).

```

00341     {
00342
00343     int varid;
00344
00345     NC(nc_inq_varid(ncid, varname, &varid));
00346     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00347 }

```



```

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

```

Definition at line 41 of file [tropo.c](#).

```

00043     {
00044
00045         ctl_t ctl;
00046
00047         met_t *met;
00048
00049         static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00050             lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];
00051
00052         static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00053             clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00054             wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];
00055
00056         static size_t count[10], start[10];
00057
00058         /* Allocate... */
00059         ALLOC(met, met_t, 1);
00060
00061         /* Check arguments... */
00062         if (argc < 4)
00063             ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00064
00065         /* Read control parameters... */
00066         read_ctl(argv[1], argc, argv, &ctl);
00067         lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00068         lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00069         dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00070         lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00071         lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00072         dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00073         h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00074
00075         /* Loop over files... */
00076         for (i = 3; i < argc; i++) {
00077
00078             /* Read meteorological data... */
00079             ctl.met_tropo = 0;
00080             if (!read_met(&ctl, argv[i], met))
00081                 continue;
00082
00083             /* Set horizontal grid... */
00084             if (!init) {
00085                 init = 1;
00086
00087                 /* Get grid... */
00088                 if (dlon <= 0)
00089                     dlon = fabs(met->lon[1] - met->lon[0]);
00090                 if (dlat <= 0)
00091                     dlat = fabs(met->lat[1] - met->lat[0]);
00092                 if (lon0 < -360 && lon1 > 360) {
00093                     lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00094                     lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00095                 }
00096                 nx = ny = 0;
00097                 for (lon = lon0; lon <= lon1; lon += dlon) {
00098                     lons[nx] = lon;
00099                     if ((++nx) > EX)
00100                         ERRMSG("Too many longitudes!");
00101                 }
00102                 if (lat0 < -90 && lat1 > 90) {
00103                     lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00104                     lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00105                 }
00106                 for (lat = lat0; lat <= lat1; lat += dlat) {
00107                     lats[ny] = lat;
00108                     if ((++ny) > EY)
00109                         ERRMSG("Too many latitudes!");
00110                 }
00111
00112                 /* Create netCDF file... */
00113                 printf("Write tropopause data file: %s\n", argv[2]);
00114                 NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00115
00116                 /* Create dimensions... */
00117                 NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00118                 NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00119                 NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00120
00121                 /* Create variables... */
00122                 NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));

```

```

00123 NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00124 NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00125 NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00126 NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00127 NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00128 if (h2o)
00129     NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00130 NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00131 NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00132 NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00133 if (h2o)
00134     NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00135 NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00136 NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00137 NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00138 if (h2o)
00139     NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00140 NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00141 NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00142 NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00143 if (h2o)
00144     NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00145
00146 /* Set attributes... */
00147 add_text_attribute(ncid, "time", "units",
00148     "seconds since 2000-01-01 00:00:00 UTC");
00149 add_text_attribute(ncid, "time", "long_name", "time");
00150 add_text_attribute(ncid, "lon", "units", "degrees_east");
00151 add_text_attribute(ncid, "lon", "long_name", "longitude");
00152 add_text_attribute(ncid, "lat", "units", "degrees_north");
00153 add_text_attribute(ncid, "lat", "long_name", "latitude");
00154
00155 add_text_attribute(ncid, "clp_z", "units", "km");
00156 add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00157 add_text_attribute(ncid, "clp_p", "units", "hPa");
00158 add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00159 add_text_attribute(ncid, "clp_t", "units", "K");
00160 add_text_attribute(ncid, "clp_t", "long_name",
00161     "cold point temperature");
00162 if (h2o) {
00163     add_text_attribute(ncid, "clp_q", "units", "ppv");
00164     add_text_attribute(ncid, "clp_q", "long_name",
00165         "cold point water vapor");
00166 }
00167
00168 add_text_attribute(ncid, "dyn_z", "units", "km");
00169 add_text_attribute(ncid, "dyn_z", "long_name",
00170     "dynamical tropopause height");
00171 add_text_attribute(ncid, "dyn_p", "units", "hPa");
00172 add_text_attribute(ncid, "dyn_p", "long_name",
00173     "dynamical tropopause pressure");
00174 add_text_attribute(ncid, "dyn_t", "units", "K");
00175 add_text_attribute(ncid, "dyn_t", "long_name",
00176     "dynamical tropopause temperature");
00177 if (h2o) {
00178     add_text_attribute(ncid, "dyn_q", "units", "ppv");
00179     add_text_attribute(ncid, "dyn_q", "long_name",
00180         "dynamical tropopause water vapor");
00181 }
00182
00183 add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00184 add_text_attribute(ncid, "wmo_1st_z", "long_name",
00185     "WMO 1st tropopause height");
00186 add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00187 add_text_attribute(ncid, "wmo_1st_p", "long_name",
00188     "WMO 1st tropopause pressure");
00189 add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00190 add_text_attribute(ncid, "wmo_1st_t", "long_name",
00191     "WMO 1st tropopause temperature");
00192 if (h2o) {
00193     add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
00194     add_text_attribute(ncid, "wmo_1st_q", "long_name",
00195         "WMO 1st tropopause water vapor");
00196 }
00197
00198 add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00199 add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00200     "WMO 2nd tropopause height");
00201 add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00202 add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00203     "WMO 2nd tropopause pressure");
00204 add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00205 add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00206     "WMO 2nd tropopause temperature");
00207 if (h2o) {
00208     add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00209     add_text_attribute(ncid, "wmo_2nd_q", "long_name",

```

```

00210                                     "WMO 2nd tropopause water vapor");
00211     }
00212
00213     /* End definition... */
00214     NC(nc_enddef(ncid));
00215
00216     /* Write longitude and latitude... */
00217     NC(nc_put_var_double(ncid, latid, lats));
00218     NC(nc_put_var_double(ncid, lonid, lons));
00219 }
00220
00221 /* Write time... */
00222 start[0] = (size_t) nt;
00223 count[0] = 1;
00224 start[1] = 0;
00225 count[1] = (size_t) ny;
00226 start[2] = 0;
00227 count[2] = (size_t) nx;
00228 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00229
00230 /* Get cold point... */
00231 ctl.met_tropo = 2;
00232 read_met_tropo(&ctl, met);
00233 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00234 for (ix = 0; ix < nx; ix++)
00235     for (iy = 0; iy < ny; iy++) {
00236         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00237                             &pt[iy * nx + ix], ci, cw, 1);
00238         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00239                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00240         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00241                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00242         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00243                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00244     }
00245
00246 /* Write data... */
00247 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00248 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00249 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00250 if (h2o)
00251     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00252
00253 /* Get dynamical tropopause... */
00254 ctl.met_tropo = 5;
00255 read_met_tropo(&ctl, met);
00256 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00257 for (ix = 0; ix < nx; ix++)
00258     for (iy = 0; iy < ny; iy++) {
00259         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00260                             &pt[iy * nx + ix], ci, cw, 1);
00261         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00262                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00263         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00264                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00265         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00266                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00267     }
00268
00269 /* Write data... */
00270 NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00271 NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00272 NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00273 if (h2o)
00274     NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00275
00276 /* Get WMO 1st tropopause... */
00277 ctl.met_tropo = 3;
00278 read_met_tropo(&ctl, met);
00279 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00280 for (ix = 0; ix < nx; ix++)
00281     for (iy = 0; iy < ny; iy++) {
00282         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00283                             &pt[iy * nx + ix], ci, cw, 1);
00284         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00285                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00286         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00287                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00288         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00289                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00290     }
00291
00292 /* Write data... */
00293 NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00294 NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00295 NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00296 if (h2o)

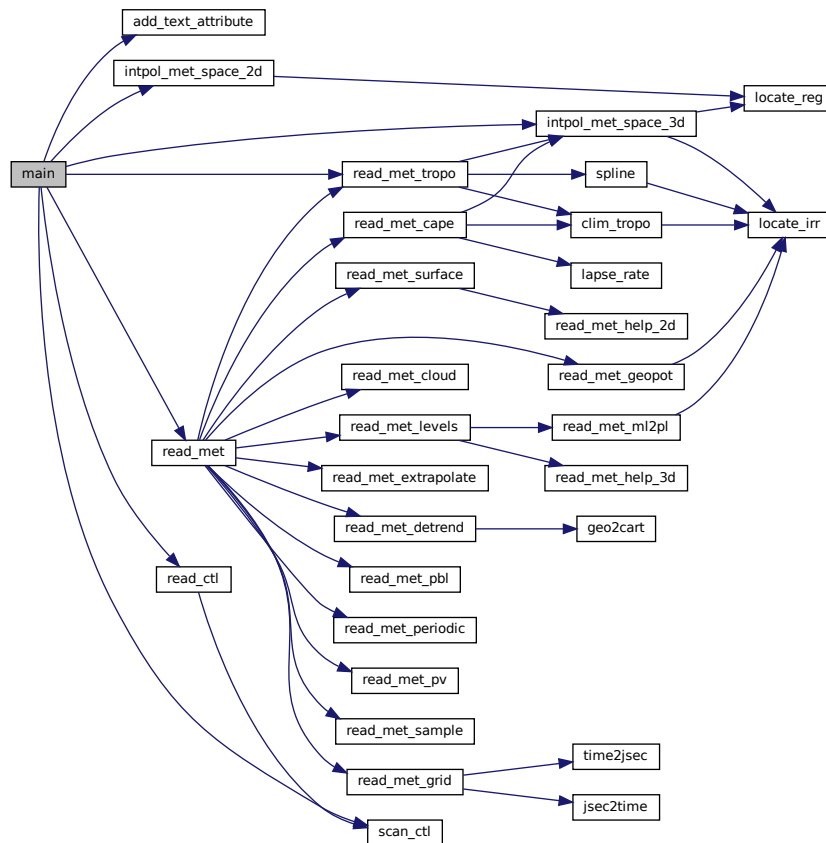
```

```

00297     NC(nc_put_vara_double(ncid, wmo1qid, start, count, qt));
00298
00299     /* Get WMO 2nd tropopause... */
00300     ctl.met_tropo = 4;
00301     read_met_tropo(&ctl, met);
00302 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00303     for (ix = 0; ix < nx; ix++)
00304     for (iy = 0; iy < ny; iy++) {
00305         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00306                             &pt[iy * nx + ix], ci, cw, 1);
00307         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00308                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00309         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00310                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00311         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00312                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00313     }
00314
00315     /* Write data... */
00316     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00317     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00318     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00319     if (h2o)
00320         NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00321
00322     /* Increment time step counter... */
00323     nt++;
00324 }
00325
00326 /* Close file... */
00327 NC(nc_close(ncid));
00328
00329 /* Free... */
00330 free(met);
00331
00332 return EXIT_SUCCESS;
00333 }

```

Here is the call graph for this function:



5.46 tropo.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) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Functions...
00029 ----- */
00030
00031 void add_text_attribute(
00032     int ncid,
00033     char *varname,
00034     char *attrname,
00035     char *text);
00036
00037 /* -----
00038  Main...
00039 ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00050         lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];
00051
00052     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00053         clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00054         wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];
00055
00056     static size_t count[10], start[10];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00068     lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00069     dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00073     h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00074
00075     /* Loop over files... */
00076     for (i = 3; i < argc; i++) {
00077
00078         /* Read meteorological data... */
00079         ctl.met_tropo = 0;
00080         if (!read_met(&ctl, argv[i], met))
00081             continue;
00082
00083         /* Set horizontal grid... */
00084         if (!init) {
00085             init = 1;
00086
00087             /* Get grid... */
00088             if (dlon <= 0)
00089                 dlon = fabs(met->lon[1] - met->lon[0]);
00090             if (dlat <= 0)

```

```

00091     dlat = fabs(met->lat[1] - met->lat[0]);
00092     if (lon0 < -360 && lon1 > 360) {
00093         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00094         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00095     }
00096     nx = ny = 0;
00097     for (lon = lon0; lon <= lon1; lon += dlon) {
00098         lons[nx] = lon;
00099         if ((++nx) > EX)
00100             ERRMSG("Too many longitudes!");
00101     }
00102     if (lat0 < -90 && lat1 > 90) {
00103         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00104         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00105     }
00106     for (lat = lat0; lat <= lat1; lat += dlat) {
00107         lats[ny] = lat;
00108         if ((++ny) > EY)
00109             ERRMSG("Too many latitudes!");
00110     }
00111
00112     /* Create netCDF file... */
00113     printf("Write tropopause data file: %s\n", argv[2]);
00114     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00115
00116     /* Create dimensions... */
00117     NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00118     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00119     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00120
00121     /* Create variables... */
00122     NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00123     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00124     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00125     NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00126     NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00127     NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00128     if (h2o)
00129         NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00130     NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00131     NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00132     NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00133     if (h2o)
00134         NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00135     NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00136     NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00137     NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00138     if (h2o)
00139         NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00140     NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00141     NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00142     NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00143     if (h2o)
00144         NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00145
00146     /* Set attributes... */
00147     add_text_attribute(ncid, "time", "units",
00148         "seconds since 2000-01-01 00:00:00 UTC");
00149     add_text_attribute(ncid, "time", "long_name", "time");
00150     add_text_attribute(ncid, "lon", "units", "degrees_east");
00151     add_text_attribute(ncid, "lon", "long_name", "longitude");
00152     add_text_attribute(ncid, "lat", "units", "degrees_north");
00153     add_text_attribute(ncid, "lat", "long_name", "latitude");
00154
00155     add_text_attribute(ncid, "clp_z", "units", "km");
00156     add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00157     add_text_attribute(ncid, "clp_p", "units", "hPa");
00158     add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00159     add_text_attribute(ncid, "clp_t", "units", "K");
00160     add_text_attribute(ncid, "clp_t", "long_name",
00161         "cold point temperature");
00162     if (h2o) {
00163         add_text_attribute(ncid, "clp_q", "units", "ppv");
00164         add_text_attribute(ncid, "clp_q", "long_name",
00165             "cold point water vapor");
00166     }
00167
00168     add_text_attribute(ncid, "dyn_z", "units", "km");
00169     add_text_attribute(ncid, "dyn_z", "long_name",
00170         "dynamical tropopause height");
00171     add_text_attribute(ncid, "dyn_p", "units", "hPa");
00172     add_text_attribute(ncid, "dyn_p", "long_name",
00173         "dynamical tropopause pressure");
00174     add_text_attribute(ncid, "dyn_t", "units", "K");
00175     add_text_attribute(ncid, "dyn_t", "long_name",
00176         "dynamical tropopause temperature");
00177     if (h2o) {

```

```

00178     add_text_attribute(ncid, "dyn_q", "units", "ppv");
00179     add_text_attribute(ncid, "dyn_q", "long_name",
00180                        "dynamical tropopause water vapor");
00181 }
00182
00183 add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00184 add_text_attribute(ncid, "wmo_1st_z", "long_name",
00185                    "WMO 1st tropopause height");
00186 add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00187 add_text_attribute(ncid, "wmo_1st_p", "long_name",
00188                    "WMO 1st tropopause pressure");
00189 add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00190 add_text_attribute(ncid, "wmo_1st_t", "long_name",
00191                    "WMO 1st tropopause temperature");
00192 if (h2o) {
00193     add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
00194     add_text_attribute(ncid, "wmo_1st_q", "long_name",
00195                        "WMO 1st tropopause water vapor");
00196 }
00197
00198 add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00199 add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00200                    "WMO 2nd tropopause height");
00201 add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00202 add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00203                    "WMO 2nd tropopause pressure");
00204 add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00205 add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00206                    "WMO 2nd tropopause temperature");
00207 if (h2o) {
00208     add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00209     add_text_attribute(ncid, "wmo_2nd_q", "long_name",
00210                        "WMO 2nd tropopause water vapor");
00211 }
00212
00213 /* End definition... */
00214 NC(nc_enddef(ncid));
00215
00216 /* Write longitude and latitude... */
00217 NC(nc_put_var_double(ncid, latid, lats));
00218 NC(nc_put_var_double(ncid, lonid, lons));
00219 }
00220
00221 /* Write time... */
00222 start[0] = (size_t) nt;
00223 count[0] = 1;
00224 start[1] = 0;
00225 count[1] = (size_t) ny;
00226 start[2] = 0;
00227 count[2] = (size_t) nx;
00228 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00229
00230 /* Get cold point... */
00231 ctl.met_tropo = 2;
00232 read_met_tropo(&ctl, met);
00233 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00234 for (ix = 0; ix < nx; ix++)
00235     for (iy = 0; iy < ny; iy++) {
00236         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00237                             &pt[iy * nx + ix], ci, cw, 1);
00238         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00239                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00240         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00241                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00242         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00243                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00244     }
00245
00246 /* Write data... */
00247 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00248 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00249 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00250 if (h2o)
00251     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00252
00253 /* Get dynamical tropopause... */
00254 ctl.met_tropo = 5;
00255 read_met_tropo(&ctl, met);
00256 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00257 for (ix = 0; ix < nx; ix++)
00258     for (iy = 0; iy < ny; iy++) {
00259         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00260                             &pt[iy * nx + ix], ci, cw, 1);
00261         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00262                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00263         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00264                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);

```

```

00265         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00266                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00267     }
00268
00269     /* Write data... */
00270     NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00271     NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00272     NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00273     if (h2o)
00274         NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00275
00276     /* Get WMO 1st tropopause... */
00277     ctl.met_tropo = 3;
00278     read_met_tropo(&ctl, met);
00279     #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00280     for (ix = 0; ix < nx; ix++)
00281         for (iy = 0; iy < ny; iy++) {
00282             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00283                               &pt[iy * nx + ix], ci, cw, 1);
00284             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00285                               lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00286             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00287                               lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00288             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00289                               lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00290         }
00291
00292     /* Write data... */
00293     NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00294     NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00295     NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00296     if (h2o)
00297         NC(nc_put_vara_double(ncid, wmolqid, start, count, qt));
00298
00299     /* Get WMO 2nd tropopause... */
00300     ctl.met_tropo = 4;
00301     read_met_tropo(&ctl, met);
00302     #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00303     for (ix = 0; ix < nx; ix++)
00304         for (iy = 0; iy < ny; iy++) {
00305             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00306                               &pt[iy * nx + ix], ci, cw, 1);
00307             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00308                               lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00309             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00310                               lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00311             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00312                               lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00313         }
00314
00315     /* Write data... */
00316     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00317     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00318     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00319     if (h2o)
00320         NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00321
00322     /* Increment time step counter... */
00323     nt++;
00324 }
00325
00326 /* Close file... */
00327 NC(nc_close(ncid));
00328
00329 /* Free... */
00330 free(met);
00331
00332 return EXIT_SUCCESS;
00333 }
00334
00335 /*****
00336
00337 void add_text_attribute(
00338     int ncid,
00339     char *varname,
00340     char *attrname,
00341     char *text) {
00342
00343     int varid;
00344
00345     NC(nc_inq_varid(ncid, varname, &varid));
00346     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00347 }

```


5.47 tropo_sample.c File Reference

```
#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], size_t nlon, size_t 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[])

5.47.1 Detailed Description

Sample tropopause data set.

Definition in file [tropo_sample.c](#).

5.47.2 Macro Definition Documentation

5.47.2.1 NT `#define NT 744`

Maximum number of time steps.

Definition at line 32 of file [tropo_sample.c](#).

5.47.3 Function Documentation

```

5.47.3.1 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],
    size_t nlon,
    size_t nlat,
    double time,
    double lon,
    double lat,
    int method,
    double * var,
    double * sigma )

```

3-D linear interpolation of tropopause data.

Definition at line 266 of file `tropo_sample.c`.

```

00280         {
00281
00282     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00283
00284     int n = 0;
00285
00286     /* Adjust longitude... */
00287     if (lon < lons[0])
00288         lon += 360;
00289     else if (lon > lons[nlon - 1])
00290         lon -= 360;
00291
00292     /* Get indices... */
00293     int ix = locate_reg(lons, (int) nlon, lon);
00294     int iy = locate_reg(lats, (int) nlat, lat);
00295
00296     /* Calculate standard deviation... */
00297     *sigma = 0;
00298     for (int dx = 0; dx < 2; dx++)
00299         for (int dy = 0; dy < 2; dy++) {
00300             if (isfinite(array0[ix + dx][iy + dy])) {
00301                 mean += array0[ix + dx][iy + dy];
00302                 *sigma += SQR(array0[ix + dx][iy + dy]);
00303                 n++;
00304             }
00305             if (isfinite(array1[ix + dx][iy + dy])) {
00306                 mean += array1[ix + dx][iy + dy];
00307                 *sigma += SQR(array1[ix + dx][iy + dy]);
00308                 n++;
00309             }
00310         }
00311     if (n > 0)
00312         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00313
00314     /* Linear interpolation... */
00315     if (method == 1 && isfinite(array0[ix][iy])
00316         && isfinite(array0[ix][iy + 1])
00317         && isfinite(array0[ix + 1][iy])
00318         && isfinite(array0[ix + 1][iy + 1])
00319         && isfinite(array1[ix][iy])
00320         && isfinite(array1[ix][iy + 1])
00321         && isfinite(array1[ix + 1][iy])
00322         && isfinite(array1[ix + 1][iy + 1])) {
00323
00324         aux00 = LIN(lons[ix], array0[ix][iy],
00325                    lons[ix + 1], array0[ix + 1][iy], lon);
00326         aux01 = LIN(lons[ix], array0[ix][iy + 1],
00327                    lons[ix + 1], array0[ix + 1][iy + 1], lon);
00328         aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00329
00330         aux10 = LIN(lons[ix], array1[ix][iy],
00331                    lons[ix + 1], array1[ix + 1][iy], lon);
00332         aux11 = LIN(lons[ix], array1[ix][iy + 1],
00333                    lons[ix + 1], array1[ix + 1][iy + 1], lon);
00334         aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00335
00336         *var = LIN(time0, aux0, time1, aux1, time);
00337     }
00338

```

```

00339  /* Nearest neighbor interpolation... */
00340  else {
00341      aux00 = NN(lons[ix], array0[ix][iy],
00342                lons[ix + 1], array0[ix + 1][iy], lon);
00343      aux01 = NN(lons[ix], array0[ix][iy + 1],
00344                lons[ix + 1], array0[ix + 1][iy + 1], lon);
00345      aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00346
00347      aux10 = NN(lons[ix], array1[ix][iy],
00348                lons[ix + 1], array1[ix + 1][iy], lon);
00349      aux11 = NN(lons[ix], array1[ix][iy + 1],
00350                lons[ix + 1], array1[ix + 1][iy + 1], lon);
00351      aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00352
00353      *var = NN(time0, aux0, time1, aux1, time);
00354  }
00355 }

```

Here is the call graph for this function:



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

Definition at line 59 of file tropo_sample.c.

```

00061      {
00062
00063      ctl_t ctl;
00064
00065      atm_t *atm;
00066
00067      static FILE *out;
00068
00069      static char varname[LEN];
00070
00071      static double times[NT], lons[EX], lats[EY], time0, time1, z0, z0sig,
00072      p0, p0sig, t0, t0sig, q0, q0sig;
00073
00074      static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00075      tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00076      tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00077
00078      static int ip, iq, it, it_old = -999, method, dimid[10], ncid,
00079      varid, varid_z, varid_p, varid_t, varid_q, h2o;
00080
00081      static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
00082
00083      /* Allocate... */
00084      ALLOC(atm, atm_t, 1);
00085
00086      /* Check arguments... */
00087      if (argc < 5)
00088          ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00089
00090      /* Read control parameters... */
00091      read_ctl(argv[1], argc, argv, &ctl);
00092      method =
00093          (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);
00094
00095      /* Read atmospheric data... */
00096      if (!read_atm(argv[5], &ctl, atm))

```

```

00097     ERRMSG("Cannot open file!");
00098
00099     /* Open tropopause file... */
00100     printf("Read tropopause data: %s\n", argv[3]);
00101     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00102         ERRMSG("Cannot open file!");
00103
00104     /* Get dimensions... */
00105     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00106     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00107     if (ntime > NT)
00108         ERRMSG("Too many times!");
00109     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00110     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00111     if (nlat > EY)
00112         ERRMSG("Too many latitudes!");
00113     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00114     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00115     if (nlon > EX)
00116         ERRMSG("Too many longitudes!");
00117
00118     /* Read coordinates... */
00119     NC(nc_inq_varid(ncid, "time", &varid));
00120     NC(nc_get_var_double(ncid, varid, times));
00121     NC(nc_inq_varid(ncid, "lat", &varid));
00122     NC(nc_get_var_double(ncid, varid, lats));
00123     NC(nc_inq_varid(ncid, "lon", &varid));
00124     NC(nc_get_var_double(ncid, varid, lons));
00125
00126     /* Get variable indices... */
00127     sprintf(varname, "%s_z", argv[4]);
00128     NC(nc_inq_varid(ncid, varname, &varid_z));
00129     sprintf(varname, "%s_p", argv[4]);
00130     NC(nc_inq_varid(ncid, varname, &varid_p));
00131     sprintf(varname, "%s_t", argv[4]);
00132     NC(nc_inq_varid(ncid, varname, &varid_t));
00133     sprintf(varname, "%s_q", argv[4]);
00134     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00135
00136     /* Set dimensions... */
00137     count[0] = 1;
00138     count[1] = nlat;
00139     count[2] = nlon;
00140
00141     /* Create file... */
00142     printf("Write tropopause sample data: %s\n", argv[2]);
00143     if (!(out = fopen(argv[2], "w")))
00144         ERRMSG("Cannot create file!");
00145
00146     /* Write header... */
00147     fprintf(out,
00148         "# $1 = time [s]\n"
00149         "# $2 = altitude [km]\n"
00150         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00151     for (iq = 0; iq < ctl.nq; iq++)
00152         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00153             ctl.qnt_unit[iq]);
00154     fprintf(out, "# $%d = tropopause height [km]\n", 5 + ctl.nq);
00155     fprintf(out, "# $%d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00156     fprintf(out, "# $%d = tropopause temperature [K]\n", 7 + ctl.nq);
00157     fprintf(out, "# $%d = tropopause water vapor [ppv]\n", 8 + ctl.nq);
00158     fprintf(out, "# $%d = tropopause height (sigma) [km]\n", 9 + ctl.nq);
00159     fprintf(out, "# $%d = tropopause pressure (sigma) [hPa]\n", 10 + ctl.nq);
00160     fprintf(out, "# $%d = tropopause temperature (sigma) [K]\n", 11 + ctl.nq);
00161     fprintf(out, "# $%d = tropopause water vapor (sigma) [ppv]\n",
00162         12 + ctl.nq);
00163
00164     /* Loop over particles... */
00165     for (ip = 0; ip < atm->np; ip++) {
00166
00167         /* Check temporal ordering... */
00168         if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00169             ERRMSG("Time must be ascending!");
00170
00171         /* Check range... */
00172         if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00173             continue;
00174
00175         /* Read data... */
00176         it = locate_irr(times, (int) ntime, atm->time[ip]);
00177         if (it != it_old) {
00178
00179             time0 = times[it];
00180             start[0] = (size_t) it;
00181             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00182             for (ilon = 0; ilon < nlon; ilon++)
00183                 for (ilat = 0; ilat < nlat; ilat++)

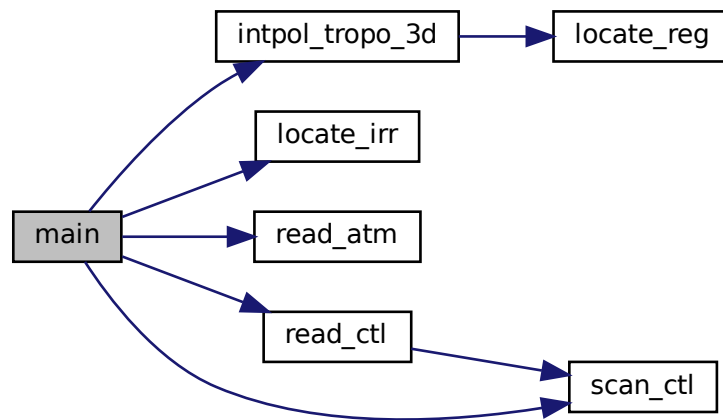
```

```

00184         tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00185     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00186     for (ilon = 0; ilon < nlon; ilon++)
00187         for (ilat = 0; ilat < nlat; ilat++)
00188             tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00189     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00190     for (ilon = 0; ilon < nlon; ilon++)
00191         for (ilat = 0; ilat < nlat; ilat++)
00192             tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00193     if (h2o) {
00194         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00195         for (ilon = 0; ilon < nlon; ilon++)
00196             for (ilat = 0; ilat < nlat; ilat++)
00197                 tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00198     } else
00199         for (ilon = 0; ilon < nlon; ilon++)
00200             for (ilat = 0; ilat < nlat; ilat++)
00201                 tropo_q0[ilon][ilat] = GSL_NAN;
00202
00203     time1 = times[it + 1];
00204     start[0] = (size_t) it + 1;
00205     NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00206     for (ilon = 0; ilon < nlon; ilon++)
00207         for (ilat = 0; ilat < nlat; ilat++)
00208             tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00209     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00210     for (ilon = 0; ilon < nlon; ilon++)
00211         for (ilat = 0; ilat < nlat; ilat++)
00212             tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00213     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00214     for (ilon = 0; ilon < nlon; ilon++)
00215         for (ilat = 0; ilat < nlat; ilat++)
00216             tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00217     if (h2o) {
00218         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00219         for (ilon = 0; ilon < nlon; ilon++)
00220             for (ilat = 0; ilat < nlat; ilat++)
00221                 tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00222     } else
00223         for (ilon = 0; ilon < nlon; ilon++)
00224             for (ilat = 0; ilat < nlat; ilat++)
00225                 tropo_q1[ilon][ilat] = GSL_NAN;;
00226 }
00227 it_old = it;
00228
00229 /* Interpolate... */
00230 intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00231                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00232                atm->lat[ip], method, &z0, &z0sig);
00233 intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00234                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00235                atm->lat[ip], method, &p0, &p0sig);
00236 intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00237                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00238                atm->lat[ip], method, &t0, &t0sig);
00239 intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00240                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00241                atm->lat[ip], method, &q0, &q0sig);
00242
00243 /* Write output... */
00244 fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00245        atm->lon[ip], atm->lat[ip]);
00246 for (iq = 0; iq < ctl.nq; iq++) {
00247     fprintf(out, " ");
00248     fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00249 }
00250 fprintf(out, " %g %g %g %g %g %g %g %g\n",
00251        z0, p0, t0, q0, z0sig, p0sig, t0sig, q0sig);
00252 }
00253
00254 /* Close files... */
00255 fclose(out);
00256 NC(nc_close(ncid));
00257
00258 /* Free... */
00259 free(atm);
00260
00261 return EXIT_SUCCESS;
00262 }

```

Here is the call graph for this function:



5.48 tropo_sample.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NT 744
00033
00034 /* -----
00035  Functions...
00036  ----- */
00037
00039 void intpol_tropo_3d(
00040     double time0,
00041     float array0[EX][EY],
00042     double time1,
00043     float array1[EX][EY],
00044     double lons[EX],
00045     double lats[EY],
00046     size_t nlon,
00047     size_t nlat,
00048     double time,
00049     double lon,
00050     double lat,
00051     int method,
00052     double *var,
00053     double *sigma);
00054
00055 /* -----
  
```

```

00056     Main...
00057     ----- */
00058
00059 int main(
00060     int argc,
00061     char *argv[]) {
00062     ctl_t ctl;
00063
00064     atm_t *atm;
00065
00066     static FILE *out;
00067
00068     static char varname[LEN];
00069
00070     static double times[NT], lons[EX], lats[EY], time0, time1, z0, z0sig,
00071         p0, p0sig, t0, t0sig, q0, q0sig;
00072
00073     static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00074         tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00075         tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00076
00077     static int ip, iq, it, it_old = -999, method, dimid[10], ncid,
00078         varid, varid_z, varid_p, varid_t, varid_q, h2o;
00079
00080     static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
00081
00082     /* Allocate... */
00083     ALLOC(atm, atm_t, 1);
00084
00085     /* Check arguments... */
00086     if (argc < 5)
00087         ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     method =
00092         (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);
00093
00094     /* Read atmospheric data... */
00095     if (!read_atm(argv[5], &ctl, atm))
00096         ERRMSG("Cannot open file!");
00097
00098     /* Open tropopause file... */
00099     printf("Read tropopause data: %s\n", argv[3]);
00100     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00101         ERRMSG("Cannot open file!");
00102
00103     /* Get dimensions... */
00104     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00105     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00106     if (ntime > NT)
00107         ERRMSG("Too many times!");
00108     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00109     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00110     if (nlat > EY)
00111         ERRMSG("Too many latitudes!");
00112     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00113     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00114     if (nlon > EX)
00115         ERRMSG("Too many longitudes!");
00116
00117     /* Read coordinates... */
00118     NC(nc_inq_varid(ncid, "time", &varid));
00119     NC(nc_get_var_double(ncid, varid, times));
00120     NC(nc_inq_varid(ncid, "lat", &varid));
00121     NC(nc_get_var_double(ncid, varid, lats));
00122     NC(nc_inq_varid(ncid, "lon", &varid));
00123     NC(nc_get_var_double(ncid, varid, lons));
00124
00125     /* Get variable indices... */
00126     sprintf(varname, "%s_z", argv[4]);
00127     NC(nc_inq_varid(ncid, varname, &varid_z));
00128     sprintf(varname, "%s_p", argv[4]);
00129     NC(nc_inq_varid(ncid, varname, &varid_p));
00130     sprintf(varname, "%s_t", argv[4]);
00131     NC(nc_inq_varid(ncid, varname, &varid_t));
00132     sprintf(varname, "%s_q", argv[4]);
00133     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00134
00135     /* Set dimensions... */
00136     count[0] = 1;
00137     count[1] = nlat;
00138     count[2] = nlon;
00139
00140     /* Create file... */
00141     printf("Write tropopause sample data: %s\n", argv[2]);

```

```

00143 if (!out = fopen(argv[2], "w"))
00144     ERRMSG("Cannot create file!");
00145
00146 /* Write header... */
00147 fprintf(out,
00148     "# $1 = time [s]\n"
00149     "# $2 = altitude [km]\n"
00150     "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00151 for (iq = 0; iq < ctl.nq; iq++)
00152     fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00153         ctl.qnt_unit[iq]);
00154 fprintf(out, "# $d = tropopause height [km]\n", 5 + ctl.nq);
00155 fprintf(out, "# $d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00156 fprintf(out, "# $d = tropopause temperature [K]\n", 7 + ctl.nq);
00157 fprintf(out, "# $d = tropopause water vapor [ppv]\n", 8 + ctl.nq);
00158 fprintf(out, "# $d = tropopause height (sigma) [km]\n", 9 + ctl.nq);
00159 fprintf(out, "# $d = tropopause pressure (sigma) [hPa]\n", 10 + ctl.nq);
00160 fprintf(out, "# $d = tropopause temperature (sigma) [K]\n", 11 + ctl.nq);
00161 fprintf(out, "# $d = tropopause water vapor (sigma) [ppv]\n",
00162     12 + ctl.nq);
00163
00164 /* Loop over particles... */
00165 for (ip = 0; ip < atm->np; ip++) {
00166
00167     /* Check temporal ordering... */
00168     if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00169         ERRMSG("Time must be ascending!");
00170
00171     /* Check range... */
00172     if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00173         continue;
00174
00175     /* Read data... */
00176     it = locate_irr(times, (int) ntime, atm->time[ip]);
00177     if (it != it_old) {
00178
00179         time0 = times[it];
00180         start[0] = (size_t) it;
00181         NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00182         for (ilon = 0; ilon < nlon; ilon++)
00183             for (ilat = 0; ilat < nlat; ilat++)
00184                 tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00185         NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00186         for (ilon = 0; ilon < nlon; ilon++)
00187             for (ilat = 0; ilat < nlat; ilat++)
00188                 tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00189         NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00190         for (ilon = 0; ilon < nlon; ilon++)
00191             for (ilat = 0; ilat < nlat; ilat++)
00192                 tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00193         if (h2o) {
00194             NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00195             for (ilon = 0; ilon < nlon; ilon++)
00196                 for (ilat = 0; ilat < nlat; ilat++)
00197                     tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00198         } else
00199             for (ilon = 0; ilon < nlon; ilon++)
00200                 for (ilat = 0; ilat < nlat; ilat++)
00201                     tropo_q0[ilon][ilat] = GSL_NAN;
00202
00203         time1 = times[it + 1];
00204         start[0] = (size_t) it + 1;
00205         NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00206         for (ilon = 0; ilon < nlon; ilon++)
00207             for (ilat = 0; ilat < nlat; ilat++)
00208                 tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00209         NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00210         for (ilon = 0; ilon < nlon; ilon++)
00211             for (ilat = 0; ilat < nlat; ilat++)
00212                 tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00213         NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00214         for (ilon = 0; ilon < nlon; ilon++)
00215             for (ilat = 0; ilat < nlat; ilat++)
00216                 tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00217         if (h2o) {
00218             NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00219             for (ilon = 0; ilon < nlon; ilon++)
00220                 for (ilat = 0; ilat < nlat; ilat++)
00221                     tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00222         } else
00223             for (ilon = 0; ilon < nlon; ilon++)
00224                 for (ilat = 0; ilat < nlat; ilat++)
00225                     tropo_q1[ilon][ilat] = GSL_NAN;
00226     }
00227     it_old = it;
00228
00229     /* Interpolate... */

```



```

00230     intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00231                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00232                    atm->lat[ip], method, &z0, &z0sig);
00233     intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00234                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00235                    atm->lat[ip], method, &p0, &p0sig);
00236     intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00237                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00238                    atm->lat[ip], method, &t0, &t0sig);
00239     intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00240                    lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00241                    atm->lat[ip], method, &q0, &q0sig);
00242
00243     /* Write output... */
00244     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00245            atm->lon[ip], atm->lat[ip]);
00246     for (iq = 0; iq < ctl.nq; iq++) {
00247         fprintf(out, " ");
00248         fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00249     }
00250     fprintf(out, " %g %g %g %g %g %g %g %g\n",
00251            z0, p0, t0, q0, z0sig, p0sig, t0sig, q0sig);
00252 }
00253
00254 /* Close files... */
00255 fclose(out);
00256 NC(nc_close(ncid));
00257
00258 /* Free... */
00259 free(atm);
00260
00261 return EXIT_SUCCESS;
00262 }
00263
00264 /*****
00265 void intpol_tropo_3d(
00266     double time0,
00267     float array0[EX][EY],
00268     double time1,
00269     float array1[EX][EY],
00270     double lons[EX],
00271     double lats[EY],
00272     size_t nlon,
00273     size_t nlat,
00274     double time,
00275     double lon,
00276     double lat,
00277     int method,
00278     double *var,
00279     double *sigma) {
00280
00281     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00282
00283     int n = 0;
00284
00285     /* Adjust longitude... */
00286     if (lon < lons[0])
00287         lon += 360;
00288     else if (lon > lons[nlon - 1])
00289         lon -= 360;
00290
00291     /* Get indices... */
00292     int ix = locate_reg(lons, (int) nlon, lon);
00293     int iy = locate_reg(lats, (int) nlat, lat);
00294
00295     /* Calculate standard deviation... */
00296     *sigma = 0;
00297     for (int dx = 0; dx < 2; dx++)
00298         for (int dy = 0; dy < 2; dy++) {
00299             if (isfinite(array0[ix + dx][iy + dy])) {
00300                 mean += array0[ix + dx][iy + dy];
00301                 *sigma += SQR(array0[ix + dx][iy + dy]);
00302                 n++;
00303             }
00304             if (isfinite(array1[ix + dx][iy + dy])) {
00305                 mean += array1[ix + dx][iy + dy];
00306                 *sigma += SQR(array1[ix + dx][iy + dy]);
00307                 n++;
00308             }
00309         }
00310     if (n > 0)
00311         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00312
00313     /* Linear interpolation... */
00314     if (method == 1 && isfinite(array0[ix][iy])
00315         && isfinite(array0[ix][iy + 1])

```

```

00317     && isfinite(array0[ix + 1][iy])
00318     && isfinite(array0[ix + 1][iy + 1])
00319     && isfinite(array1[ix][iy])
00320     && isfinite(array1[ix][iy + 1])
00321     && isfinite(array1[ix + 1][iy])
00322     && isfinite(array1[ix + 1][iy + 1])) {
00323
00324     aux00 = LIN(lons[ix], array0[ix][iy],
00325                lons[ix + 1], array0[ix + 1][iy], lon);
00326     aux01 = LIN(lons[ix], array0[ix][iy + 1],
00327                lons[ix + 1], array0[ix + 1][iy + 1], lon);
00328     aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00329
00330     aux10 = LIN(lons[ix], array1[ix][iy],
00331                lons[ix + 1], array1[ix + 1][iy], lon);
00332     aux11 = LIN(lons[ix], array1[ix][iy + 1],
00333                lons[ix + 1], array1[ix + 1][iy + 1], lon);
00334     aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00335
00336     *var = LIN(time0, aux0, time1, aux1, time);
00337 }
00338
00339 /* Nearest neighbor interpolation... */
00340 else {
00341     aux00 = NN(lons[ix], array0[ix][iy],
00342               lons[ix + 1], array0[ix + 1][iy], lon);
00343     aux01 = NN(lons[ix], array0[ix][iy + 1],
00344               lons[ix + 1], array0[ix + 1][iy + 1], lon);
00345     aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00346
00347     aux10 = NN(lons[ix], array1[ix][iy],
00348               lons[ix + 1], array1[ix + 1][iy], lon);
00349     aux11 = NN(lons[ix], array1[ix][iy + 1],
00350               lons[ix + 1], array1[ix + 1][iy + 1], lon);
00351     aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00352
00353     *var = NN(time0, aux0, time1, aux1, time);
00354 }
00355 }

```

5.49 wind.c File Reference

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

Functions

- void [add_text_attribute](#) (int ncid, char *varname, char *attrname, char *text)
- int [main](#) (int argc, char *argv[])

5.49.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file [wind.c](#).

5.49.2 Function Documentation

5.49.2.1 add_text_attribute() void add_text_attribute (

```

    int ncid,
    char * varname,
    char * attrname,
    char * text )

```

Definition at line 188 of file [wind.c](#).

```

00192     {
00193
00194     int varid;
00195
00196     NC(nc_inq_varid(ncid, varname, &varid));
00197     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00198 }

```

5.49.2.2 main() int main (

```

    int argc,
    char * argv[] )

```

Definition at line 41 of file [wind.c](#).

```

00043     {
00044
00045     ctl_t ctl;
00046
00047     static char filename[LEN];
00048
00049     static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050         u0, u1, alpha;
00051
00052     static float *dataT, *dataU, *dataV, *dataW;
00053
00054     static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055         idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
00057     /* Allocate... */
00058     ALLOC(dataT, float,
00059         EP * EY * EX);
00060     ALLOC(dataU, float,
00061         EP * EY * EX);
00062     ALLOC(dataV, float,
00063         EP * EY * EX);
00064     ALLOC(dataW, float,
00065         EP * EY * EX);
00066
00067     /* Check arguments... */
00068     if (argc < 3)
00069         ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071     /* Read control parameters... */
00072     read_ctl(argv[1], argc, argv, &ctl);
00073     t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00074     nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00075     ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00076     nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00077     z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00078     z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00079     u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00080     u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00081     alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00082
00083     /* Check dimensions... */
00084     if (nx < 1 || nx > EX)
00085         ERRMSG("Set 1 <= NX <= MAX!");
00086     if (ny < 1 || ny > EY)
00087         ERRMSG("Set 1 <= NY <= MAX!");
00088     if (nz < 1 || nz > EP)
00089         ERRMSG("Set 1 <= NZ <= MAX!");
00090
00091     /* Get time... */
00092     jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00093     t0 = year * 10000. + mon * 100. + day + hour / 24.;
00094
00095     /* Set filename... */
00096     sprintf(filename, "%s_%d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00097
00098     /* Create netCDF file... */

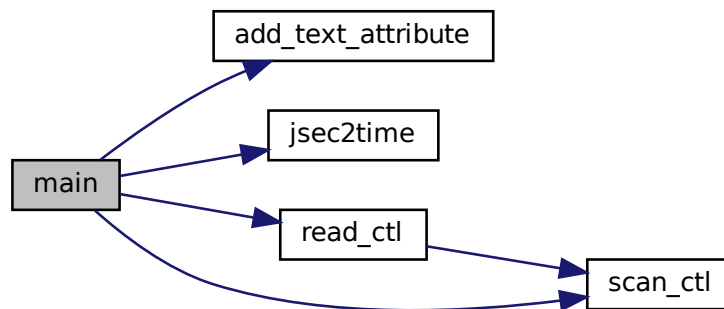
```

```

00099 NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
00101 /* Create dimensions... */
00102 NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00103 NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00104 NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00105 NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00106
00107 /* Create variables... */
00108 NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00109 NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00110 NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00111 NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00112 NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00113 NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00114 NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00115 NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00116
00117 /* Set attributes... */
00118 add_text_attribute(ncid, "time", "long_name", "time");
00119 add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
00120 add_text_attribute(ncid, "lon", "long_name", "longitude");
00121 add_text_attribute(ncid, "lon", "units", "degrees_east");
00122 add_text_attribute(ncid, "lat", "long_name", "latitude");
00123 add_text_attribute(ncid, "lat", "units", "degrees_north");
00124 add_text_attribute(ncid, "lev", "long_name", "air_pressure");
00125 add_text_attribute(ncid, "lev", "units", "Pa");
00126 add_text_attribute(ncid, "T", "long_name", "Temperature");
00127 add_text_attribute(ncid, "T", "units", "K");
00128 add_text_attribute(ncid, "U", "long_name", "U velocity");
00129 add_text_attribute(ncid, "U", "units", "m s**-1");
00130 add_text_attribute(ncid, "V", "long_name", "V velocity");
00131 add_text_attribute(ncid, "V", "units", "m s**-1");
00132 add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00133 add_text_attribute(ncid, "W", "units", "Pa s**-1");
00134
00135 /* End definition... */
00136 NC(nc_enddef(ncid));
00137
00138 /* Set coordinates... */
00139 for (ix = 0; ix < nx; ix++)
00140     dataLon[ix] = 360.0 / nx * (double) ix;
00141 for (iy = 0; iy < ny; iy++)
00142     dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00143 for (iz = 0; iz < nz; iz++)
00144     dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00145
00146 /* Write coordinates... */
00147 NC(nc_put_var_double(ncid, timid, &t0));
00148 NC(nc_put_var_double(ncid, levid, dataZ));
00149 NC(nc_put_var_double(ncid, lonid, dataLon));
00150 NC(nc_put_var_double(ncid, latid, dataLat));
00151
00152 /* Create wind fields (Williamson et al., 1992)... */
00153 for (ix = 0; ix < nx; ix++)
00154     for (iy = 0; iy < ny; iy++)
00155         for (iz = 0; iz < nz; iz++) {
00156             idx = (iz * ny + iy) * nx + ix;
00157             dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00158                 * (cos(dataLat[iy] * M_PI / 180.0)
00159                 * cos(alpha * M_PI / 180.0)
00160                 + sin(dataLat[iy] * M_PI / 180.0)
00161                 * cos(dataLon[ix] * M_PI / 180.0)
00162                 * sin(alpha * M_PI / 180.0)));
00163             dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00164                 * sin(dataLon[ix] * M_PI / 180.0)
00165                 * sin(alpha * M_PI / 180.0));
00166         }
00167
00168 /* Write wind data... */
00169 NC(nc_put_var_float(ncid, tid, dataT));
00170 NC(nc_put_var_float(ncid, uid, dataU));
00171 NC(nc_put_var_float(ncid, vid, dataV));
00172 NC(nc_put_var_float(ncid, wid, dataW));
00173
00174 /* Close file... */
00175 NC(nc_close(ncid));
00176
00177 /* Free... */
00178 free(dataT);
00179 free(dataU);
00180 free(dataV);
00181 free(dataW);
00182
00183 return EXIT_SUCCESS;
00184 }

```

Here is the call graph for this function:



5.50 wind.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2021 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Functions...
00029 ----- */
00030
00031 void add_text_attribute(
00032     int ncid,
00033     char *varname,
00034     char *attrname,
00035     char *text);
00036
00037 /* -----
00038  Main...
00039 ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     static char filename[LEN];
00048
00049     static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050         u0, u1, alpha;
00051
00052     static float *dataT, *dataU, *dataV, *dataW;
00053
00054     static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055         idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
00057     /* Allocate... */
00058     ALLOC(dataT, float,

```

```

00059     EP * EY * EX);
00060     ALLOC(dataU, float,
00061           EP * EY * EX);
00062     ALLOC(dataV, float,
00063           EP * EY * EX);
00064     ALLOC(dataW, float,
00065           EP * EY * EX);
00066
00067     /* Check arguments... */
00068     if (argc < 3)
00069         ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071     /* Read control parameters... */
00072     read_ctl(argv[1], argc, argv, &ctl);
00073     t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00074     nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00075     ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00076     nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00077     z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00078     z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00079     u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00080     u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00081     alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00082
00083     /* Check dimensions... */
00084     if (nx < 1 || nx > EX)
00085         ERRMSG("Set 1 <= NX <= MAX!");
00086     if (ny < 1 || ny > EY)
00087         ERRMSG("Set 1 <= NY <= MAX!");
00088     if (nz < 1 || nz > EP)
00089         ERRMSG("Set 1 <= NZ <= MAX!");
00090
00091     /* Get time... */
00092     jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00093     t0 = year * 10000. + mon * 100. + day + hour / 24.;
00094
00095     /* Set filename... */
00096     sprintf(filename, "%s_%d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00097
00098     /* Create netCDF file... */
00099     NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
00101     /* Create dimensions... */
00102     NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00103     NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00104     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00105     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00106
00107     /* Create variables... */
00108     NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00109     NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00110     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00111     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00112     NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00113     NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00114     NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00115     NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00116
00117     /* Set attributes... */
00118     add_text_attribute(ncid, "time", "long_name", "time");
00119     add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
00120     add_text_attribute(ncid, "lon", "long_name", "longitude");
00121     add_text_attribute(ncid, "lon", "units", "degrees_east");
00122     add_text_attribute(ncid, "lat", "long_name", "latitude");
00123     add_text_attribute(ncid, "lat", "units", "degrees_north");
00124     add_text_attribute(ncid, "lev", "long_name", "air_pressure");
00125     add_text_attribute(ncid, "lev", "units", "Pa");
00126     add_text_attribute(ncid, "T", "long_name", "Temperature");
00127     add_text_attribute(ncid, "T", "units", "K");
00128     add_text_attribute(ncid, "U", "long_name", "U velocity");
00129     add_text_attribute(ncid, "U", "units", "m s**-1");
00130     add_text_attribute(ncid, "V", "long_name", "V velocity");
00131     add_text_attribute(ncid, "V", "units", "m s**-1");
00132     add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00133     add_text_attribute(ncid, "W", "units", "Pa s**-1");
00134
00135     /* End definition... */
00136     NC(nc_enddef(ncid));
00137
00138     /* Set coordinates... */
00139     for (ix = 0; ix < nx; ix++)
00140         dataLon[ix] = 360.0 / nx * (double) ix;
00141     for (iy = 0; iy < ny; iy++)
00142         dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00143     for (iz = 0; iz < nz; iz++)
00144         dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00145

```

```

00146  /* Write coordinates... */
00147  NC(nc_put_var_double(ncid, timid, &t0));
00148  NC(nc_put_var_double(ncid, levid, dataZ));
00149  NC(nc_put_var_double(ncid, lonid, dataLon));
00150  NC(nc_put_var_double(ncid, latid, dataLat));
00151
00152  /* Create wind fields (Williamson et al., 1992)... */
00153  for (ix = 0; ix < nx; ix++)
00154      for (iy = 0; iy < ny; iy++)
00155          for (iz = 0; iz < nz; iz++) {
00156              idx = (iz * ny + iy) * nx + ix;
00157              dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00158                                   * (cos(dataLat[iy] * M_PI / 180.0)
00159                                       * cos(alpha * M_PI / 180.0)
00160                                           + sin(dataLat[iy] * M_PI / 180.0)
00161                                               * cos(dataLon[ix] * M_PI / 180.0)
00162                                                   * sin(alpha * M_PI / 180.0)));
00163              dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00164                                    * sin(dataLon[ix] * M_PI / 180.0)
00165                                        * sin(alpha * M_PI / 180.0));
00166          }
00167
00168  /* Write wind data... */
00169  NC(nc_put_var_float(ncid, tid, dataT));
00170  NC(nc_put_var_float(ncid, uid, dataU));
00171  NC(nc_put_var_float(ncid, vid, dataV));
00172  NC(nc_put_var_float(ncid, wid, dataW));
00173
00174  /* Close file... */
00175  NC(nc_close(ncid));
00176
00177  /* Free... */
00178  free(dataT);
00179  free(dataU);
00180  free(dataV);
00181  free(dataW);
00182
00183  return EXIT_SUCCESS;
00184 }
00185
00186 /*****
00187
00188 void add_text_attribute(
00189     int ncid,
00190     char *varname,
00191     char *attrname,
00192     char *text) {
00193
00194     int varid;
00195
00196     NC(nc_inq_varid(ncid, varname, &varid));
00197     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00198 }

```


Index

add_text_attribute
 tropo.c, [412](#)
 wind.c, [430](#)
advect
 ctl_t, [28](#)
ALLOC
 libtrac.h, [227](#)
atm_basename
 ctl_t, [33](#)
atm_conv.c, [50](#), [51](#)
 main, [51](#)
atm_dist.c, [52](#), [57](#)
 main, [52](#)
atm_dt_out
 ctl_t, [33](#)
atm_filter
 ctl_t, [33](#)
atm_gpfile
 ctl_t, [33](#)
atm_init.c, [62](#), [64](#)
 main, [62](#)
atm_select.c, [65](#), [68](#)
 main, [66](#)
atm_split.c, [70](#), [72](#)
 main, [70](#)
atm_stat.c, [74](#), [77](#)
 main, [74](#)
atm_stride
 ctl_t, [34](#)
atm_t, [2](#)
 lat, [3](#)
 lon, [3](#)
 np, [3](#)
 p, [3](#)
 q, [3](#)
 time, [3](#)
atm_type
 ctl_t, [34](#)

balloon
 ctl_t, [27](#)
bound_dps
 ctl_t, [31](#)
bound_lat0
 ctl_t, [30](#)
bound_lat1
 ctl_t, [31](#)
bound_mass
 ctl_t, [30](#)
bound_p0
 ctl_t, [31](#)
bound_p1
 ctl_t, [31](#)
bound_vmr
 ctl_t, [30](#)

cache_t, [4](#)
 iso_n, [5](#)
 iso_ps, [5](#)
 iso_ts, [5](#)
 iso_var, [5](#)
 tsig, [4](#)
 uvwp, [5](#)
 uvwsig, [4](#)
cape
 met_t, [48](#)
cart2geo
 libtrac.c, [87](#)
 libtrac.h, [239](#)
check_finite
 libtrac.h, [240](#)
chunkszhint
 ctl_t, [13](#)
cin
 met_t, [48](#)
cl
 met_t, [47](#)
clim_hno3
 libtrac.c, [87](#)
 libtrac.h, [240](#)
clim_oh
 libtrac.c, [88](#)
 libtrac.h, [241](#)
clim_tropo
 libtrac.c, [89](#)
 libtrac.h, [241](#)
conv_cape
 ctl_t, [29](#)
conv_cin
 ctl_t, [29](#)
conv_dt
 ctl_t, [30](#)
conv_mix_bot
 ctl_t, [30](#)
conv_mix_top
 ctl_t, [30](#)
conv_wcape
 ctl_t, [29](#)
conv_wmax
 ctl_t, [29](#)
CPD
 libtrac.h, [223](#)
csi_basename
 ctl_t, [34](#)
csi_dt_out
 ctl_t, [34](#)
csi_lat0
 ctl_t, [36](#)
csi_lat1
 ctl_t, [36](#)
csi_lon0

ctl_t, 35
 csi_lon1
 ctl_t, 36
 csi_modmin
 ctl_t, 35
 csi_nx
 ctl_t, 35
 csi_ny
 ctl_t, 36
 csi_nz
 ctl_t, 35
 csi_obsfile
 ctl_t, 34
 csi_obsmin
 ctl_t, 34
 csi_z0
 ctl_t, 35
 csi_z1
 ctl_t, 35
 ctl_t, 6
 advect, 28
 atm_basename, 33
 atm_dt_out, 33
 atm_filter, 33
 atm_gpfile, 33
 atm_stride, 34
 atm_type, 34
 balloon, 27
 bound_dps, 31
 bound_lat0, 30
 bound_lat1, 31
 bound_mass, 30
 bound_p0, 31
 bound_p1, 31
 bound_vmr, 30
 chunkszhint, 13
 conv_cape, 29
 conv_cin, 29
 conv_dt, 30
 conv_mix_bot, 30
 conv_mix_top, 30
 conv_wcape, 29
 conv_wmax, 29
 csi_basename, 34
 csi_dt_out, 34
 csi_lat0, 36
 csi_lat1, 36
 csi_lon0, 35
 csi_lon1, 36
 csi_modmin, 35
 csi_nx, 35
 csi_ny, 36
 csi_nz, 35
 csi_obsfile, 34
 csi_obsmin, 34
 csi_z0, 35
 csi_z1, 35
 direction, 23
 dry_depo, 32
 dt_met, 23
 dt_mod, 23
 ens_basename, 40
 grid_basename, 36
 grid_dt_out, 37
 grid_gpfile, 36
 grid_lat0, 38
 grid_lat1, 38
 grid_lon0, 38
 grid_lon1, 38
 grid_nx, 37
 grid_ny, 38
 grid_nz, 37
 grid_sparse, 37
 grid_z0, 37
 grid_z1, 37
 isosurf, 27
 met_cache, 27
 met_cloud, 27
 met_detrend, 25
 met_dp, 24
 met_dt_out, 27
 met_dx, 24
 met_dy, 24
 met_geopot_sx, 25
 met_geopot_sy, 25
 met_np, 25
 met_p, 25
 met_sp, 24
 met_sx, 24
 met_sy, 24
 met_tropo, 25
 met_tropo_lapse, 26
 met_tropo_lapse_sep, 26
 met_tropo_nlev, 26
 met_tropo_nlev_sep, 26
 met_tropo_pv, 26
 met_tropo_spline, 27
 met_tropo_theta, 26
 metbase, 23
 molmass, 31
 nq, 13
 oh_chem, 32
 oh_chem_reaction, 32
 prof_basename, 38
 prof_lat0, 40
 prof_lat1, 40
 prof_lon0, 39
 prof_lon1, 40
 prof_nx, 39
 prof_ny, 40
 prof_nz, 39
 prof_obsfile, 39
 prof_z0, 39
 prof_z1, 39
 psc_h2o, 33
 psc_hno3, 33

qnt_cape, 19
qnt_cin, 19
qnt_cl, 18
qnt_ens, 14
qnt_format, 14
qnt_h2o, 17
qnt_h2ot, 16
qnt_hno3, 19
qnt_iwc, 18
qnt_lapse, 21
qnt_lwc, 18
qnt_m, 14
qnt_name, 13
qnt_o3, 18
qnt_oh, 20
qnt_p, 17
qnt_pbl, 16
qnt_pcb, 18
qnt_pct, 18
qnt_pel, 19
qnt_plcl, 19
qnt_plfc, 19
qnt_ps, 15
qnt_psat, 20
qnt_psice, 20
qnt_pt, 16
qnt_pv, 22
qnt_pw, 20
qnt_r, 15
qnt_rh, 20
qnt_rhice, 21
qnt_rho, 14
qnt_sh, 20
qnt_stat, 14
qnt_t, 17
qnt_tdew, 22
qnt_theta, 21
qnt_tice, 22
qnt_tnat, 22
qnt_ts, 15
qnt_tsts, 22
qnt_tt, 16
qnt_tvirt, 21
qnt_u, 17
qnt_unit, 13
qnt_us, 15
qnt_v, 17
qnt_vh, 21
qnt_vmr, 14
qnt_vs, 15
qnt_vz, 22
qnt_w, 17
qnt_z, 16
qnt_zeta, 21
qnt_zs, 15
qnt_zt, 16
read_mode, 13
reflect, 28
sample_basename, 40
sample_dx, 41
sample_dz, 41
sample_obsfile, 41
species, 31
stat_basename, 41
stat_lat, 41
stat_lon, 41
stat_r, 42
stat_t0, 42
stat_t1, 42
t_start, 23
t_stop, 23
tdec_strat, 32
tdec_trop, 32
turb_dx_strat, 28
turb_dx_trop, 28
turb_dz_strat, 28
turb_dz_trop, 28
turb_mesox, 29
turb_mesoz, 29
wet_depo, 32
day2doy
 libtrac.c, 90
 libtrac.h, 242
day2doy.c, 80, 81
 main, 80
DEG2DX
 libtrac.h, 227
DEG2DY
 libtrac.h, 228
direction
 ctl_t, 23
DIST
 libtrac.h, 229
DIST2
 libtrac.h, 229
DLAPSE
 met_lapse.c, 319
DOTP
 libtrac.h, 229
doy2day
 libtrac.c, 90
 libtrac.h, 242
doy2day.c, 82, 83
 main, 82
DP2DZ
 libtrac.h, 228
dry_depo
 ctl_t, 32
dt_met
 ctl_t, 23
dt_mod
 ctl_t, 23
DX2DEG
 libtrac.h, 228
DY2DEG
 libtrac.h, 228

DZ2DP
 libtrac.h, 228
 ens_basename
 ctl_t, 40
 EP
 libtrac.h, 226
 EPS
 libtrac.h, 223
 ERRMSG
 libtrac.h, 238
 EX
 libtrac.h, 226
 EY
 libtrac.h, 226
 fft_help
 met_spec.c, 345
 FMOD
 libtrac.h, 229
 FREAD
 libtrac.h, 229
 FWRITE
 libtrac.h, 230
 G0
 libtrac.h, 223
 geo2cart
 libtrac.c, 90
 libtrac.h, 243
 get_met
 libtrac.c, 91
 libtrac.h, 243
 get_met_help
 libtrac.c, 93
 libtrac.h, 245
 get_met_replace
 libtrac.c, 94
 libtrac.h, 246
 grid_basename
 ctl_t, 36
 grid_dt_out
 ctl_t, 37
 grid_gpfile
 ctl_t, 36
 grid_lat0
 ctl_t, 38
 grid_lat1
 ctl_t, 38
 grid_lon0
 ctl_t, 38
 grid_lon1
 ctl_t, 38
 grid_nx
 ctl_t, 37
 grid_ny
 ctl_t, 38
 grid_nz
 ctl_t, 37
 grid_sparse
 ctl_t, 37
 grid_z0
 ctl_t, 37
 grid_z1
 ctl_t, 37
 GX
 libtrac.h, 226
 GY
 libtrac.h, 227
 GZ
 libtrac.h, 227
 H0
 libtrac.h, 224
 h2o
 met_t, 49
 h2ot
 met_t, 47
 IDXMAX
 met_lapse.c, 319
 INTPOL_2D
 libtrac.h, 230
 INTPOL_3D
 libtrac.h, 230
 INTPOL_INIT
 libtrac.h, 230
 intpol_met_space_2d
 libtrac.c, 95
 libtrac.h, 248
 intpol_met_space_3d
 libtrac.c, 94
 libtrac.h, 247
 intpol_met_time_2d
 libtrac.c, 97
 libtrac.h, 250
 intpol_met_time_3d
 libtrac.c, 96
 libtrac.h, 249
 INTPOL_SPACE_ALL
 libtrac.h, 231
 INTPOL_TIME_ALL
 libtrac.h, 231
 intpol_tropo_3d
 tropo_sample.c, 421
 iso_n
 cache_t, 5
 iso_ps
 cache_t, 5
 iso_ts
 cache_t, 5
 iso_var
 cache_t, 5
 isosurf
 ctl_t, 27
 iwc
 met_t, 50

- jsec2time
 - libtrac.c, 98
 - libtrac.h, 250
- jsec2time.c, 83, 84
 - main, 83
- KB
 - libtrac.h, 224
- LAPSE
 - libtrac.h, 232
- lapse_rate
 - libtrac.c, 98
 - libtrac.h, 251
- LAPSEMIN
 - met_lapse.c, 319
- lat
 - atm_t, 3
 - met_t, 45
- LEN
 - libtrac.h, 225
- libtrac.c, 85, 151
 - cart2geo, 87
 - clim_hno3, 87
 - clim_oh, 88
 - clim_tropo, 89
 - day2doy, 90
 - doy2day, 90
 - geo2cart, 90
 - get_met, 91
 - get_met_help, 93
 - get_met_replace, 94
 - intpol_met_space_2d, 95
 - intpol_met_space_3d, 94
 - intpol_met_time_2d, 97
 - intpol_met_time_3d, 96
 - jsec2time, 98
 - lapse_rate, 98
 - locate_irr, 99
 - locate_reg, 99
 - nat_temperature, 100
 - read_atm, 100
 - read_ctl, 103
 - read_met, 108
 - read_met_cape, 110
 - read_met_cloud, 112
 - read_met_detrend, 112
 - read_met_extrapolate, 114
 - read_met_geopot, 114
 - read_met_grid, 116
 - read_met_help_2d, 119
 - read_met_help_3d, 118
 - read_met_levels, 121
 - read_met_ml2pl, 122
 - read_met_pbl, 123
 - read_met_periodic, 124
 - read_met_pv, 124
 - read_met_sample, 126
 - read_met_surface, 127
 - read_met_tropo, 128
 - scan_ctl, 131
 - sedi, 132
 - spline, 132
 - stddev, 133
 - time2jsec, 134
 - timer, 134
 - tropo_weight, 135
 - write_atm, 136
 - write_csi, 138
 - write_ens, 141
 - write_grid, 142
 - write_prof, 145
 - write_sample, 148
 - write_station, 150
- libtrac.h, 217, 303
 - ALLOC, 227
 - cart2geo, 239
 - check_finite, 240
 - clim_hno3, 240
 - clim_oh, 241
 - clim_tropo, 241
 - CPD, 223
 - day2doy, 242
 - DEG2DX, 227
 - DEG2DY, 228
 - DIST, 229
 - DIST2, 229
 - DOTP, 229
 - doy2day, 242
 - DP2DZ, 228
 - DX2DEG, 228
 - DY2DEG, 228
 - DZ2DP, 228
 - EP, 226
 - EPS, 223
 - ERRMSG, 238
 - EX, 226
 - EY, 226
 - FMOD, 229
 - FREAD, 229
 - FWRITE, 230
 - G0, 223
 - geo2cart, 243
 - get_met, 243
 - get_met_help, 245
 - get_met_replace, 246
 - GX, 226
 - GY, 227
 - GZ, 227
 - H0, 224
 - INTPOL_2D, 230
 - INTPOL_3D, 230
 - INTPOL_INIT, 230
 - intpol_met_space_2d, 248
 - intpol_met_space_3d, 247
 - intpol_met_time_2d, 250
 - intpol_met_time_3d, 249

INTPOL_SPACE_ALL, 231
 INTPOL_TIME_ALL, 231
 jsec2time, 250
 KB, 224
 LAPSE, 232
 lapse_rate, 251
 LEN, 225
 LIN, 232
 locate_irr, 251
 locate_reg, 252
 LOG, 237
 LOGLEV, 237
 LV, 224
 MA, 224
 MH2O, 224
 MO3, 224
 nat_temperature, 252
 NC, 232
 NENS, 227
 NN, 233
 NORM, 233
 NP, 226
 NQ, 226
 NTHREADS, 227
 NTIMER, 238
 NVTX_POP, 239
 NVTX_PUSH, 239
 P, 233
 P0, 225
 PRINT, 238
 PRINT_TIMERS, 238
 PSAT, 233
 PSICE, 233
 PW, 234
 RA, 225
 RE, 225
 read_atm, 252
 read_ctl, 255
 read_met, 261
 read_met_cape, 262
 read_met_cloud, 264
 read_met_detrend, 265
 read_met_extrapolate, 266
 read_met_geopot, 267
 read_met_grid, 268
 read_met_help_2d, 272
 read_met_help_3d, 270
 read_met_levels, 273
 read_met_ml2pl, 274
 read_met_pbl, 275
 read_met_periodic, 276
 read_met_pv, 277
 read_met_sample, 278
 read_met_surface, 280
 read_met_tropo, 280
 RH, 234
 RHICE, 234
 RI, 225
 scan_ctl, 283
 sedi, 284
 SELECT_TIMER, 238
 SET_ATM, 234
 SET_QNT, 234
 SH, 235
 spline, 284
 SQR, 235
 START_TIMERS, 239
 stddev, 285
 STOP_TIMERS, 239
 T0, 225
 TDEW, 235
 THETA, 235
 THETA_VIRT, 236
 TICE, 235
 time2jsec, 286
 timer, 286
 TOK, 236
 tropo_weight, 287
 TVIRT, 236
 WARN, 237
 write_atm, 288
 write_csi, 290
 write_ens, 293
 write_grid, 294
 write_prof, 297
 write_sample, 300
 write_station, 302
 Z, 236
 ZDIFF, 236
 ZETA, 237
 LIN
 libtrac.h, 232
 locate_irr
 libtrac.c, 99
 libtrac.h, 251
 locate_reg
 libtrac.c, 99
 libtrac.h, 252
 LOG
 libtrac.h, 237
 LOGLEV
 libtrac.h, 237
 lon
 atm_t, 3
 met_t, 45
 LV
 libtrac.h, 224
 lwc
 met_t, 50
 MA
 libtrac.h, 224
 main
 atm_conv.c, 51
 atm_dist.c, 52
 atm_init.c, 62
 atm_select.c, 66

- atm_split.c, 70
- atm_stat.c, 74
- day2doy.c, 80
- doy2day.c, 82
- jsec2time.c, 83
- met_lapse.c, 319
- met_map.c, 326
- met_prof.c, 333
- met_sample.c, 340
- met_spec.c, 345
- met_subgrid.c, 350
- met_zm.c, 355
- sedi.c, 363
- time2jsec.c, 365
- tnat.c, 366
- trac.c, 389
- tropo.c, 412
- tropo_sample.c, 423
- wind.c, 431
- met_cache
 - ctl_t, 27
- met_cloud
 - ctl_t, 27
- met_detrend
 - ctl_t, 25
- met_dp
 - ctl_t, 24
- met_dt_out
 - ctl_t, 27
- met_dx
 - ctl_t, 24
- met_dy
 - ctl_t, 24
- met_geopot_sx
 - ctl_t, 25
- met_geopot_sy
 - ctl_t, 25
- met_lapse.c, 318, 322
 - DLAPSE, 319
 - IDXMAX, 319
 - LAPSEMIN, 319
 - main, 319
- met_map.c, 325, 329
 - main, 326
 - NX, 325
 - NY, 325
- met_np
 - ctl_t, 25
- met_p
 - ctl_t, 25
- met_prof.c, 332, 336
 - main, 333
 - NZ, 333
- met_sample.c, 339, 342
 - main, 340
- met_sp
 - ctl_t, 24
- met_spec.c, 344, 347
 - fft_help, 345
 - main, 345
 - PMAX, 345
- met_subgrid.c, 349, 352
 - main, 350
- met_sx
 - ctl_t, 24
- met_sy
 - ctl_t, 24
- met_t, 42
 - cape, 48
 - cin, 48
 - cl, 47
 - h2o, 49
 - h2ot, 47
 - iwc, 50
 - lat, 45
 - lon, 45
 - lwc, 50
 - np, 45
 - nx, 44
 - ny, 44
 - o3, 49
 - p, 45
 - pbl, 46
 - pcb, 47
 - pct, 47
 - pel, 48
 - pl, 50
 - plcl, 47
 - plfc, 48
 - ps, 45
 - pt, 46
 - pv, 49
 - t, 48
 - time, 44
 - ts, 45
 - tt, 46
 - u, 49
 - us, 46
 - v, 49
 - vs, 46
 - w, 49
 - z, 48
 - zs, 46
 - zt, 47
- met_tropo
 - ctl_t, 25
- met_tropo_lapse
 - ctl_t, 26
- met_tropo_lapse_sep
 - ctl_t, 26
- met_tropo_nlev
 - ctl_t, 26
- met_tropo_nlev_sep
 - ctl_t, 26
- met_tropo_pv
 - ctl_t, 26

met_tropo_spline
 ctl_t, 27
 met_tropo_theta
 ctl_t, 26
 met_zm.c, 355, 359
 main, 355
 NY, 355
 NZ, 355
 metbase
 ctl_t, 23
 MH2O
 libtrac.h, 224
 MO3
 libtrac.h, 224
 module_advect_mp
 trac.c, 369
 module_advect_rk
 trac.c, 370
 module_bound_cond
 trac.c, 371
 module_convection
 trac.c, 371
 module_decay
 trac.c, 373
 module_diffusion_meso
 trac.c, 373
 module_diffusion_turb
 trac.c, 375
 module_dry_deposition
 trac.c, 376
 module_isosurf
 trac.c, 378
 module_isosurf_init
 trac.c, 377
 module_meteo
 trac.c, 379
 module_oh_chem
 trac.c, 381
 module_position
 trac.c, 382
 module_rng
 trac.c, 383
 module_rng_init
 trac.c, 383
 module_sedi
 trac.c, 384
 module_timesteps
 trac.c, 385
 module_timesteps_init
 trac.c, 385
 module_wet_deposition
 trac.c, 386
 molmass
 ctl_t, 31
 nat_temperature
 libtrac.c, 100
 libtrac.h, 252
 NC
 libtrac.h, 232
 NENS
 libtrac.h, 227
 NN
 libtrac.h, 233
 NORM
 libtrac.h, 233
 NP
 libtrac.h, 226
 np
 atm_t, 3
 met_t, 45
 NQ
 libtrac.h, 226
 nq
 ctl_t, 13
 NT
 tropo_sample.c, 421
 NTHREADS
 libtrac.h, 227
 NTIMER
 libtrac.h, 238
 NVTX_POP
 libtrac.h, 239
 NVTX_PUSH
 libtrac.h, 239
 NX
 met_map.c, 325
 nx
 met_t, 44
 NY
 met_map.c, 325
 met_zm.c, 355
 ny
 met_t, 44
 NZ
 met_prof.c, 333
 met_zm.c, 355

 o3
 met_t, 49
 oh_chem
 ctl_t, 32
 oh_chem_reaction
 ctl_t, 32

 P
 libtrac.h, 233
 p
 atm_t, 3
 met_t, 45
 P0
 libtrac.h, 225
 pbl
 met_t, 46
 pcb
 met_t, 47
 pct
 met_t, 47

pel
 met_t, 48
pl
 met_t, 50
plcl
 met_t, 47
plfc
 met_t, 48
PMAX
 met_spec.c, 345
PRINT
 libtrac.h, 238
PRINT_TIMERS
 libtrac.h, 238
prof_basename
 ctl_t, 38
prof_lat0
 ctl_t, 40
prof_lat1
 ctl_t, 40
prof_lon0
 ctl_t, 39
prof_lon1
 ctl_t, 40
prof_nx
 ctl_t, 39
prof_ny
 ctl_t, 40
prof_nz
 ctl_t, 39
prof_obsfile
 ctl_t, 39
prof_z0
 ctl_t, 39
prof_z1
 ctl_t, 39
ps
 met_t, 45
PSAT
 libtrac.h, 233
psc_h2o
 ctl_t, 33
psc_hno3
 ctl_t, 33
PSICE
 libtrac.h, 233
pt
 met_t, 46
pv
 met_t, 49
PW
 libtrac.h, 234
q
 atm_t, 3
qnt_cape
 ctl_t, 19
qnt_cin
 ctl_t, 19
qnt_cl
 ctl_t, 18
qnt_ens
 ctl_t, 14
qnt_format
 ctl_t, 14
qnt_h2o
 ctl_t, 17
qnt_h2ot
 ctl_t, 16
qnt_hno3
 ctl_t, 19
qnt_iwc
 ctl_t, 18
qnt_lapse
 ctl_t, 21
qnt_lwc
 ctl_t, 18
qnt_m
 ctl_t, 14
qnt_name
 ctl_t, 13
qnt_o3
 ctl_t, 18
qnt_oh
 ctl_t, 20
qnt_p
 ctl_t, 17
qnt_pbl
 ctl_t, 16
qnt_pcb
 ctl_t, 18
qnt_pct
 ctl_t, 18
qnt_pel
 ctl_t, 19
qnt_plcl
 ctl_t, 19
qnt_plfc
 ctl_t, 19
qnt_ps
 ctl_t, 15
qnt_psat
 ctl_t, 20
qnt_psice
 ctl_t, 20
qnt_pt
 ctl_t, 16
qnt_pv
 ctl_t, 22
qnt_pw
 ctl_t, 20
qnt_r
 ctl_t, 15
qnt_rh
 ctl_t, 20
qnt_rhice
 ctl_t, 21

qnt_rho
 ctl_t, 14
 qnt_sh
 ctl_t, 20
 qnt_stat
 ctl_t, 14
 qnt_t
 ctl_t, 17
 qnt_tdew
 ctl_t, 22
 qnt_theta
 ctl_t, 21
 qnt_tice
 ctl_t, 22
 qnt_tnat
 ctl_t, 22
 qnt_ts
 ctl_t, 15
 qnt_tsts
 ctl_t, 22
 qnt_tt
 ctl_t, 16
 qnt_tvirt
 ctl_t, 21
 qnt_u
 ctl_t, 17
 qnt_unit
 ctl_t, 13
 qnt_us
 ctl_t, 15
 qnt_v
 ctl_t, 17
 qnt_vh
 ctl_t, 21
 qnt_vmr
 ctl_t, 14
 qnt_vs
 ctl_t, 15
 qnt_vz
 ctl_t, 22
 qnt_w
 ctl_t, 17
 qnt_z
 ctl_t, 16
 qnt_zeta
 ctl_t, 21
 qnt_zs
 ctl_t, 15
 qnt_zt
 ctl_t, 16

 RA
 libtrac.h, 225
 RE
 libtrac.h, 225
 read_atm
 libtrac.c, 100
 libtrac.h, 252
 read_ctl
 libtrac.c, 103
 libtrac.h, 255
 read_met
 libtrac.c, 108
 libtrac.h, 261
 read_met_cape
 libtrac.c, 110
 libtrac.h, 262
 read_met_cloud
 libtrac.c, 112
 libtrac.h, 264
 read_met_detrend
 libtrac.c, 112
 libtrac.h, 265
 read_met_extrapolate
 libtrac.c, 114
 libtrac.h, 266
 read_met_geopot
 libtrac.c, 114
 libtrac.h, 267
 read_met_grid
 libtrac.c, 116
 libtrac.h, 268
 read_met_help_2d
 libtrac.c, 119
 libtrac.h, 272
 read_met_help_3d
 libtrac.c, 118
 libtrac.h, 270
 read_met_levels
 libtrac.c, 121
 libtrac.h, 273
 read_met_ml2pl
 libtrac.c, 122
 libtrac.h, 274
 read_met_pbl
 libtrac.c, 123
 libtrac.h, 275
 read_met_periodic
 libtrac.c, 124
 libtrac.h, 276
 read_met_pv
 libtrac.c, 124
 libtrac.h, 277
 read_met_sample
 libtrac.c, 126
 libtrac.h, 278
 read_met_surface
 libtrac.c, 127
 libtrac.h, 280
 read_met_tropo
 libtrac.c, 128
 libtrac.h, 280
 read_mode
 ctl_t, 13
 reflect
 ctl_t, 28
 RH

- libtrac.h, 234
- RHICE
 - libtrac.h, 234
- RI
 - libtrac.h, 225
- sample_basename
 - ctl_t, 40
- sample_dx
 - ctl_t, 41
- sample_dz
 - ctl_t, 41
- sample_obsfile
 - ctl_t, 41
- scan_ctl
 - libtrac.c, 131
 - libtrac.h, 283
- sedi
 - libtrac.c, 132
 - libtrac.h, 284
- sedi.c, 362, 364
 - main, 363
- SELECT_TIMER
 - libtrac.h, 238
- SET_ATM
 - libtrac.h, 234
- SET_QNT
 - libtrac.h, 234
- SH
 - libtrac.h, 235
- species
 - ctl_t, 31
- spline
 - libtrac.c, 132
 - libtrac.h, 284
- SQR
 - libtrac.h, 235
- START_TIMERS
 - libtrac.h, 239
- stat_basename
 - ctl_t, 41
- stat_lat
 - ctl_t, 41
- stat_lon
 - ctl_t, 41
- stat_r
 - ctl_t, 42
- stat_t0
 - ctl_t, 42
- stat_t1
 - ctl_t, 42
- stddev
 - libtrac.c, 133
 - libtrac.h, 285
- STOP_TIMERS
 - libtrac.h, 239
- t
 - met_t, 48
- T0
 - libtrac.h, 225
- t_start
 - ctl_t, 23
- t_stop
 - ctl_t, 23
- tdec_strat
 - ctl_t, 32
- tdec_trop
 - ctl_t, 32
- TDEW
 - libtrac.h, 235
- THETA
 - libtrac.h, 235
- THETAVIRT
 - libtrac.h, 236
- TICE
 - libtrac.h, 235
- time
 - atm_t, 3
 - met_t, 44
- time2jsec
 - libtrac.c, 134
 - libtrac.h, 286
- time2jsec.c, 364, 365
 - main, 365
- timer
 - libtrac.c, 134
 - libtrac.h, 286
- tnat.c, 366, 367
 - main, 366
- TOK
 - libtrac.h, 236
- trac.c, 368, 393
 - main, 389
 - module_advect_mp, 369
 - module_advect_rk, 370
 - module_bound_cond, 371
 - module_convection, 371
 - module_decay, 373
 - module_diffusion_meso, 373
 - module_diffusion_turb, 375
 - module_dry_deposition, 376
 - module_isosurf, 378
 - module_isosurf_init, 377
 - module_meteo, 379
 - module_oh_chem, 381
 - module_position, 382
 - module_rng, 383
 - module_rng_init, 383
 - module_sedi, 384
 - module_timesteps, 385
 - module_timesteps_init, 385
 - module_wet_deposition, 386
 - write_output, 387
- tropo.c, 412, 417
 - add_text_attribute, 412
 - main, 412

tropo_sample.c, [421](#), [426](#)
 intpol_tropo_3d, [421](#)
 main, [423](#)
 NT, [421](#)
tropo_weight
 libtrac.c, [135](#)
 libtrac.h, [287](#)
ts
 met_t, [45](#)
tsig
 cache_t, [4](#)
tt
 met_t, [46](#)
turb_dx_strat
 ctl_t, [28](#)
turb_dx_trop
 ctl_t, [28](#)
turb_dz_strat
 ctl_t, [28](#)
turb_dz_trop
 ctl_t, [28](#)
turb_mesox
 ctl_t, [29](#)
turb_mesoz
 ctl_t, [29](#)
TVIRT
 libtrac.h, [236](#)

u
 met_t, [49](#)
us
 met_t, [46](#)
uvwp
 cache_t, [5](#)
uvwsig
 cache_t, [4](#)

v
 met_t, [49](#)
vs
 met_t, [46](#)

w
 met_t, [49](#)
WARN
 libtrac.h, [237](#)
wet_depo
 ctl_t, [32](#)
wind.c, [430](#), [433](#)
 add_text_attribute, [430](#)
 main, [431](#)
write_atm
 libtrac.c, [136](#)
 libtrac.h, [288](#)
write_csi
 libtrac.c, [138](#)
 libtrac.h, [290](#)
write_ens
 libtrac.c, [141](#)

 libtrac.h, [293](#)
write_grid
 libtrac.c, [142](#)
 libtrac.h, [294](#)
write_output
 trac.c, [387](#)
write_prof
 libtrac.c, [145](#)
 libtrac.h, [297](#)
write_sample
 libtrac.c, [148](#)
 libtrac.h, [300](#)
write_station
 libtrac.c, [150](#)
 libtrac.h, [302](#)

Z
 libtrac.h, [236](#)
z
 met_t, [48](#)
ZDIFF
 libtrac.h, [236](#)
ZETA
 libtrac.h, [237](#)
zs
 met_t, [46](#)
zt
 met_t, [47](#)