

MPTRAC

Generated by Doxygen 1.9.1

1 Main Page	1
2 Data Structure Index	1
2.1 Data Structures	1
3 File Index	2
3.1 File List	2
4 Data Structure Documentation	3
4.1 atm_t Struct Reference	3
4.1.1 Detailed Description	4
4.1.2 Field Documentation	4
4.2 cache_t Struct Reference	5
4.2.1 Detailed Description	5
4.2.2 Field Documentation	6
4.3 clim_t Struct Reference	6
4.3.1 Detailed Description	8
4.3.2 Field Documentation	8
4.4 ctl_t Struct Reference	12
4.4.1 Detailed Description	21
4.4.2 Field Documentation	21
4.5 met_t Struct Reference	58
4.5.1 Detailed Description	60
4.5.2 Field Documentation	60
5 File Documentation	67
5.1 atm_conv.c File Reference	67
5.1.1 Detailed Description	67
5.1.2 Function Documentation	67
5.2 atm_conv.c	68
5.3 atm_dist.c File Reference	69
5.3.1 Detailed Description	69
5.3.2 Function Documentation	70
5.4 atm_dist.c	74
5.5 atm_init.c File Reference	79
5.5.1 Detailed Description	79
5.5.2 Function Documentation	79
5.6 atm_init.c	81
5.7 atm_select.c File Reference	83
5.7.1 Detailed Description	83
5.7.2 Function Documentation	83
5.8 atm_select.c	86
5.9 atm_split.c File Reference	87
5.9.1 Detailed Description	87

5.9.2 Function Documentation	88
5.10 atm_split.c	90
5.11 atm_stat.c File Reference	92
5.11.1 Detailed Description	92
5.11.2 Function Documentation	92
5.12 atm_stat.c	95
5.13 cape.c File Reference	98
5.13.1 Detailed Description	98
5.13.2 Function Documentation	98
5.14 cape.c	100
5.15 day2doy.c File Reference	102
5.15.1 Detailed Description	102
5.15.2 Function Documentation	102
5.16 day2doy.c	103
5.17 doy2day.c File Reference	104
5.17.1 Detailed Description	104
5.17.2 Function Documentation	104
5.18 doy2day.c	105
5.19 jsec2time.c File Reference	105
5.19.1 Detailed Description	105
5.19.2 Function Documentation	105
5.20 jsec2time.c	106
5.21 libtrac.c File Reference	107
5.21.1 Detailed Description	110
5.21.2 Function Documentation	110
5.22 libtrac.c	209
5.23 libtrac.h File Reference	288
5.23.1 Detailed Description	296
5.23.2 Macro Definition Documentation	296
5.23.3 Function Documentation	316
5.24 libtrac.h	415
5.25 met_conv.c File Reference	436
5.25.1 Detailed Description	437
5.25.2 Function Documentation	437
5.26 met_conv.c	438
5.27 met_lapse.c File Reference	439
5.27.1 Detailed Description	440
5.27.2 Macro Definition Documentation	440
5.27.3 Function Documentation	440
5.28 met_lapse.c	444
5.29 met_map.c File Reference	447
5.29.1 Detailed Description	447

5.29.2 Macro Definition Documentation	448
5.29.3 Function Documentation	448
5.30 met_map.c	452
5.31 met_prof.c File Reference	456
5.31.1 Detailed Description	456
5.31.2 Macro Definition Documentation	456
5.31.3 Function Documentation	456
5.32 met_prof.c	460
5.33 met_sample.c File Reference	463
5.33.1 Detailed Description	464
5.33.2 Function Documentation	464
5.34 met_sample.c	467
5.35 met_spec.c File Reference	470
5.35.1 Detailed Description	470
5.35.2 Macro Definition Documentation	470
5.35.3 Function Documentation	470
5.36 met_spec.c	473
5.37 met_subgrid.c File Reference	475
5.37.1 Detailed Description	476
5.37.2 Function Documentation	476
5.38 met_subgrid.c	479
5.39 met_zm.c File Reference	482
5.39.1 Detailed Description	482
5.39.2 Macro Definition Documentation	482
5.39.3 Function Documentation	483
5.40 met_zm.c	486
5.41 sedi.c File Reference	490
5.41.1 Detailed Description	490
5.41.2 Function Documentation	490
5.42 sedi.c	491
5.43 time2jsec.c File Reference	492
5.43.1 Detailed Description	492
5.43.2 Function Documentation	492
5.44 time2jsec.c	493
5.45 tnat.c File Reference	493
5.45.1 Detailed Description	494
5.45.2 Function Documentation	494
5.46 tnat.c	495
5.47 trac.c File Reference	495
5.47.1 Detailed Description	496
5.47.2 Function Documentation	496
5.48 trac.c	527

5.49 tropo.c File Reference	552
5.49.1 Detailed Description	552
5.49.2 Function Documentation	552
5.50 tropo.c	557
5.51 tropo_sample.c File Reference	561
5.51.1 Detailed Description	561
5.51.2 Macro Definition Documentation	562
5.51.3 Function Documentation	562
5.52 tropo_sample.c	566
5.53 tropo_zm.c File Reference	571
5.53.1 Detailed Description	571
5.53.2 Macro Definition Documentation	571
5.53.3 Function Documentation	571
5.54 tropo_zm.c	574
5.55 wind.c File Reference	577
5.55.1 Detailed Description	577
5.55.2 Function Documentation	577
5.56 wind.c	579
Index	583

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	3
cache_t	
Cache data	5
clim_t	
Climatological data	6
ctl_t	
Control parameters	12

met_t	
Meteo data	58

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

atm_conv.c	
Convert file format of air parcel data files	67
atm_dist.c	
Calculate transport deviations of trajectories	69
atm_init.c	
Create atmospheric data file with initial air parcel positions	79
atm_select.c	
Extract subsets of air parcels from atmospheric data files	83
atm_split.c	
Split air parcels into a larger number of parcels	87
atm_stat.c	
Calculate air parcel statistics	92
cape.c	
Add CAPE data to netCDF file	98
day2doy.c	
Convert date to day of year	102
doy2day.c	
Convert day of year to date	104
jsec2time.c	
Convert Julian seconds to date	105
libtrac.c	
MPTRAC library definitions	107
libtrac.h	
MPTRAC library declarations	288
met_conv.c	
Convert file format of meteo data files	436
met_lapse.c	
Calculate lapse rate statistics	439
met_map.c	
Extract map from meteorological data	447
met_prof.c	
Extract vertical profile from meteorological data	456

met_sample.c	Sample meteorological data at given geolocations	463
met_spec.c	Spectral analysis of meteorological data	470
met_subgrid.c	Calculate standard deviations of horizontal wind and vertical velocity	475
met_zm.c	Extract zonal mean from meteorological data	482
sedi.c	Calculate sedimentation velocity	490
time2jsec.c	Convert date to Julian seconds	492
tnat.c	Calculate PSC temperatures	493
trac.c	Lagrangian particle dispersion model	495
tropo.c	Create tropopause data set from meteorological data	552
tropo_sample.c	Sample tropopause data set	561
tropo_zm.c	Extract zonal mean of tropopause data set	571
wind.c	Create meteorological data files with synthetic wind fields	577

4 Data Structure Documentation

4.1 atm_t Struct Reference

Atmospheric data.

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

Data Fields

- int [np](#)
Number of air parcels.
- double [time](#) [NP]
Time [s].
- double [p](#) [NP]
Pressure [hPa].
- double [zeta](#) [NP]
Zeta [K].

- double `lon` [NP]
Longitude [deg].
- double `lat` [NP]
Latitude [deg].
- double `q` [NQ][NP]
Quantity data (for various, user-defined attributes).

4.1.1 Detailed Description

Atmospheric data.

Definition at line 1373 of file `libtrac.h`.

4.1.2 Field Documentation

4.1.2.1 `np` `int atm_t::np`

Number of air parcels.

Definition at line 1376 of file `libtrac.h`.

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

Time [s].

Definition at line 1379 of file `libtrac.h`.

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

Pressure [hPa].

Definition at line 1382 of file `libtrac.h`.

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

Zeta [K].

Definition at line 1385 of file `libtrac.h`.

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

Longitude [deg].

Definition at line 1388 of file [libtrac.h](#).

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

Latitude [deg].

Definition at line 1391 of file [libtrac.h](#).

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

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

Definition at line 1394 of file [libtrac.h](#).

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

- [libtrac.h](#)

4.2 `cache_t` Struct Reference

Cache data.

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

Data Fields

- `double iso_var[NP]`
Isosurface variables.
- `double iso_ps[NP]`
Isosurface balloon pressure [hPa].
- `double iso_ts[NP]`
Isosurface balloon time [s].
- `int iso_n`
Isosurface balloon number of data points.
- `float uvwp[NP][3]`
Wind perturbations [m/s].

4.2.1 Detailed Description

Cache data.

Definition at line 1399 of file [libtrac.h](#).

4.2.2 Field Documentation

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

Isosurface variables.

Definition at line [1402](#) of file [libtrac.h](#).

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

Isosurface balloon pressure [hPa].

Definition at line [1405](#) of file [libtrac.h](#).

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

Isosurface balloon time [s].

Definition at line [1408](#) of file [libtrac.h](#).

4.2.2.4 **iso_n** `int cache_t::iso_n`

Isosurface balloon number of data points.

Definition at line [1411](#) of file [libtrac.h](#).

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

Wind perturbations [m/s].

Definition at line [1414](#) of file [libtrac.h](#).

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

- [libtrac.h](#)

4.3 **clim_t** Struct Reference

Climatological data.

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

Data Fields

- int [tropo_ftime](#)
Number of tropopause timesteps.
- int [tropo_nlat](#)
Number of tropopause latitudes.
- double [tropo_time](#) [12]
Tropopause time steps [s].
- double [tropo_lat](#) [73]
Tropopause latitudes [deg].
- double [tropo](#) [12][73]
Tropopause pressure values [hPa].
- int [hno3_ftime](#)
Number of HNO3 timesteps.
- int [hno3_nlat](#)
Number of HNO3 latitudes.
- int [hno3_np](#)
Number of HNO3 pressure levels.
- double [hno3_time](#) [12]
HNO3 time steps [s].
- double [hno3_lat](#) [18]
HNO3 latitudes [deg].
- double [hno3_p](#) [10]
HNO3 pressure levels [hPa].
- double [hno3](#) [12][18][10]
HNO3 volume mixing ratios [ppv].
- int [oh_ftime](#)
Number of OH timesteps.
- int [oh_nlat](#)
Number of OH latitudes.
- int [oh_np](#)
Number of OH pressure levels.
- double [oh_time](#) [CT]
OH time steps [s].
- double [oh_lat](#) [CY]
OH latitudes [deg].
- double [oh_p](#) [CP]
OH pressure levels [hPa].
- double [oh](#) [CT][CP][CY]
OH number concentrations [molec/cm³].
- int [h2o2_ftime](#)
Number of H2O2 timesteps.
- int [h2o2_nlat](#)
Number of H2O2 latitudes.
- int [h2o2_np](#)
Number of H2O2 pressure levels.
- double [h2o2_time](#) [CT]
H2O2 time steps [s].
- double [h2o2_lat](#) [CY]
H2O2 latitudes [deg].
- double [h2o2_p](#) [CP]
H2O2 pressure levels [hPa].
- double [h2o2](#) [CT][CP][CY]
H2O2 number concentrations [molec/cm³].

4.3.1 Detailed Description

Climatological data.

Definition at line [1419](#) of file [libtrac.h](#).

4.3.2 Field Documentation

4.3.2.1 **tropo_ntime** `int clim_t::tropo_ntime`

Number of tropopause timesteps.

Definition at line [1422](#) of file [libtrac.h](#).

4.3.2.2 **tropo_nlat** `int clim_t::tropo_nlat`

Number of tropopause latitudes.

Definition at line [1425](#) of file [libtrac.h](#).

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

Tropopause time steps [s].

Definition at line [1428](#) of file [libtrac.h](#).

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

Tropopause latitudes [deg].

Definition at line [1431](#) of file [libtrac.h](#).

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

Tropopause pressure values [hPa].

Definition at line [1434](#) of file [libtrac.h](#).

4.3.2.6 hno3_ntime `int clim_t::hno3_ntime`

Number of HNO3 timesteps.

Definition at line [1437](#) of file [libtrac.h](#).

4.3.2.7 hno3_nlat `int clim_t::hno3_nlat`

Number of HNO3 latitudes.

Definition at line [1440](#) of file [libtrac.h](#).

4.3.2.8 hno3_np `int clim_t::hno3_np`

Number of HNO3 pressure levels.

Definition at line [1443](#) of file [libtrac.h](#).

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

HNO3 time steps [s].

Definition at line [1446](#) of file [libtrac.h](#).

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

HNO3 latitudes [deg].

Definition at line [1449](#) of file [libtrac.h](#).

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

HNO3 pressure levels [hPa].

Definition at line [1452](#) of file [libtrac.h](#).

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

HNO3 volume mixing ratios [ppv].

Definition at line [1455](#) of file [libtrac.h](#).

4.3.2.13 oh_ntime `int clim_t::oh_ntime`

Number of OH timesteps.

Definition at line [1458](#) of file [libtrac.h](#).

4.3.2.14 oh_nlat `int clim_t::oh_nlat`

Number of OH latitudes.

Definition at line [1461](#) of file [libtrac.h](#).

4.3.2.15 oh_np `int clim_t::oh_np`

Number of OH pressure levels.

Definition at line [1464](#) of file [libtrac.h](#).

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

OH time steps [s].

Definition at line [1467](#) of file [libtrac.h](#).

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

OH latitudes [deg].

Definition at line [1470](#) of file [libtrac.h](#).

4.3.2.18 oh_p double clim_t::oh_p[CP]

OH pressure levels [hPa].

Definition at line 1473 of file libtrac.h.

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

OH number concentrations [molec/cm³].

Definition at line 1476 of file libtrac.h.

4.3.2.20 h2o2_ntime int clim_t::h2o2_ntime

Number of H2O2 timesteps.

Definition at line 1479 of file libtrac.h.

4.3.2.21 h2o2_nlat int clim_t::h2o2_nlat

Number of H2O2 latitudes.

Definition at line 1482 of file libtrac.h.

4.3.2.22 h2o2_np int clim_t::h2o2_np

Number of H2O2 pressure levels.

Definition at line 1485 of file libtrac.h.

4.3.2.23 h2o2_time double clim_t::h2o2_time[CT]

H2O2 time steps [s].

Definition at line 1488 of file libtrac.h.

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

H2O2 latitudes [deg].

Definition at line 1491 of file [libtrac.h](#).

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

H2O2 pressure levels [hPa].

Definition at line 1494 of file [libtrac.h](#).

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

H2O2 number concentrations [molec/cm³].

Definition at line 1497 of file [libtrac.h](#).

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

- [libtrac.h](#)

4.4 ctl_t Struct Reference

Control parameters.

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

Data Fields

- int [vert_coord_ap](#)
Vertical coordinate of air parcels (0=pressure, 1=zeta).
- int [vert_coord_met](#)
Vertical coordinate of input meteo data (0=automatic, 1=eta).
- int [vert_vel](#)
Vertical velocity (0=kinematic, 1=diabatic).
- int [clams_met_data](#)
Read MPTRAC or CLaMS meteo data (0=MPTRAC, 1=CLaMS).
- size_t [chunkszhint](#)
Chunk size hint for nc__open.
- int [read_mode](#)
Read mode for nc__open.
- int [nq](#)
Number of quantities.
- char [qnt_name](#) [NQ][LEN]
Quantity names.

- char `qnt_longname` [NQ][LEN]
Quantity long names.
- char `qnt_unit` [NQ][LEN]
Quantity units.
- char `qnt_format` [NQ][LEN]
Quantity output format.
- int `qnt_idx`
Quantity array index for air parcel IDs.
- int `qnt_ens`
Quantity array index for ensemble IDs.
- int `qnt_stat`
Quantity array index for station flag.
- int `qnt_m`
Quantity array index for mass.
- int `qnt_vmr`
Quantity array index for volume mixing ratio.
- int `qnt_rp`
Quantity array index for particle radius.
- int `qnt_rhop`
Quantity array index for particle density.
- int `qnt_ps`
Quantity array index for surface pressure.
- int `qnt_ts`
Quantity array index for surface temperature.
- int `qnt_zs`
Quantity array index for surface geopotential height.
- int `qnt_us`
Quantity array index for surface zonal wind.
- int `qnt_vs`
Quantity array index for surface meridional wind.
- int `qnt_pbl`
Quantity array index for boundary layer pressure.
- int `qnt_pt`
Quantity array index for tropopause pressure.
- int `qnt_tt`
Quantity array index for tropopause temperature.
- int `qnt_zt`
Quantity array index for tropopause geopotential height.
- int `qnt_h2ot`
Quantity array index for tropopause water vapor vmr.
- int `qnt_z`
Quantity array index for geopotential height.
- int `qnt_p`
Quantity array index for pressure.
- int `qnt_t`
Quantity array index for temperature.
- int `qnt_rho`
Quantity array index for density of air.
- int `qnt_u`
Quantity array index for zonal wind.
- int `qnt_v`

- Quantity array index for meridional wind.*
- int [qnt_w](#)
Quantity array index for vertical velocity.
- int [qnt_h2o](#)
Quantity array index for water vapor vmr.
- int [qnt_o3](#)
Quantity array index for ozone vmr.
- int [qnt_lwc](#)
Quantity array index for cloud liquid water content.
- int [qnt_iwc](#)
Quantity array index for cloud ice water content.
- int [qnt_pct](#)
Quantity array index for cloud top pressure.
- int [qnt_pcb](#)
Quantity array index for cloud bottom pressure.
- int [qnt_cl](#)
Quantity array index for total column cloud water.
- int [qnt_plcl](#)
Quantity array index for pressure at lifted condensation level (LCL).
- int [qnt_plfc](#)
Quantity array index for pressure at level of free convection (LCF).
- int [qnt_pel](#)
Quantity array index for pressure at equilibrium level (EL).
- int [qnt_cape](#)
Quantity array index for convective available potential energy (CAPE).
- int [qnt_cin](#)
Quantity array index for convective inhibition (CIN).
- int [qnt_hno3](#)
Quantity array index for nitric acid vmr.
- int [qnt_oh](#)
Quantity array index for hydroxyl number concentrations.
- int [qnt_vmrimpl](#)
Quantity array index for implicity volumn mixing ratio.
- int [qnt_mloss_oh](#)
Quantity array index for total mass loss due to OH chemistry.
- int [qnt_mloss_h2o2](#)
Quantity array index for total mass loss due to H2O2 chemistry.
- int [qnt_mloss_wet](#)
Quantity array index for total mass loss due to wet deposition.
- int [qnt_mloss_dry](#)
Quantity array index for total mass loss due to dry deposition.
- int [qnt_mloss_decay](#)
Quantity array index for total mass loss due to exponential decax.
- int [qnt_psat](#)
Quantity array index for saturation pressure over water.
- int [qnt_psice](#)
Quantity array index for saturation pressure over ice.
- int [qnt_pw](#)
Quantity array index for partial water vapor pressure.
- int [qnt_sh](#)
Quantity array index for specific humidity.

- `int qnt_rh`
Quantity array index for relative humidity over water.
- `int qnt_rhice`
Quantity array index for relative humidity over ice.
- `int qnt_theta`
Quantity array index for potential temperature.
- `int qnt_zeta`
Quantity array index for zeta vertical coordinate.
- `int qnt_tvirt`
Quantity array index for virtual temperature.
- `int qnt_lapse`
Quantity array index for lapse rate.
- `int qnt_vh`
Quantity array index for horizontal wind.
- `int qnt_vz`
Quantity array index for vertical velocity.
- `int qnt_pv`
Quantity array index for potential vorticity.
- `int qnt_tdew`
Quantity array index for dew point temperature.
- `int qnt_tice`
Quantity array index for T_{ice} .
- `int qnt_tsts`
Quantity array index for T_{STS} .
- `int qnt_tnat`
Quantity array index for T_{NAT} .
- `int direction`
Direction flag (1=forward calculation, -1=backward calculation).
- `double t_start`
Start time of simulation [s].
- `double t_stop`
Stop time of simulation [s].
- `double dt_mod`
Time step of simulation [s].
- `char metbase [LEN]`
Basename for meteo data.
- `double dt_met`
Time step of meteo data [s].
- `int met_type`
Type of meteo data files (0=netCDF, 1=binary, 2=pack, 3=zfp, 4=zstd).
- `int met_nc_scale`
Check netCDF scaling factors (0=no, 1=yes).
- `int met_dx`
Stride for longitudes.
- `int met_dy`
Stride for latitudes.
- `int met_dp`
Stride for pressure levels.
- `int met_sx`
Smoothing for longitudes.
- `int met_sy`

- Smoothing for latitudes.*

 - int `met_sp`
- Smoothing for pressure levels.*

 - double `met_detrend`

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

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

Target pressure levels [hPa].
- int `met_geopot_sx`

Longitudinal smoothing of geopotential heights.
- int `met_geopot_sy`

Latitudinal smoothing of geopotential heights.
- int `met_relhum`

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

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

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

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

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

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

Dynamical tropopause potential vorticity threshold [PVU].
- double `met_tropo_theta`

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

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

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

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

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

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

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

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

Balloon position filename.
- int `advect`

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

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

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

- double `turb_dx_strat`
Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].
- double `turb_dz_trop`
Vertical turbulent diffusion coefficient (troposphere) [m^2/s].
- double `turb_dz_strat`
Vertical turbulent diffusion coefficient (stratosphere) [m^2/s].
- double `turb_mesox`
Horizontal scaling factor for mesoscale wind fluctuations.
- double `turb_mesoz`
Vertical scaling factor for mesoscale wind fluctuations.
- double `conv_cape`
CAPE threshold for convection module [J/kg].
- double `conv_cin`
CIN threshold for convection module [J/kg].
- double `conv_dt`
Time interval for convection module [s].
- int `conv_mix`
Type of vertical mixing (0=pressure, 1=density).
- int `conv_mix_bot`
Lower level for mixing (0=particle pressure, 1=surface).
- int `conv_mix_top`
Upper level for mixing (0=particle pressure, 1=EL).
- double `bound_mass`
Boundary conditions mass per particle [kg].
- double `bound_mass_trend`
Boundary conditions mass per particle trend [kg/s].
- double `bound_vmr`
Boundary conditions volume mixing ratio [ppv].
- double `bound_vmr_trend`
Boundary conditions volume mixing ratio trend [ppv/s].
- double `bound_lat0`
Boundary conditions minimum longitude [deg].
- double `bound_lat1`
Boundary conditions maximum longitude [deg].
- double `bound_p0`
Boundary conditions bottom pressure [hPa].
- double `bound_p1`
Boundary conditions top pressure [hPa].
- double `bound_dps`
Boundary conditions surface layer depth [hPa].
- double `bound_dzs`
Boundary conditions surface layer depth [km].
- double `bound_zetas`
Boundary conditions surface layer zeta [K].
- int `bound_pbl`
Boundary conditions planetary boundary layer (0=no, 1=yes).
- char `species` [LEN]
Species.
- double `molmass`
Molar mass [g/mol].
- double `tdec_trop`

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

 - double `tdec_strat`
- Life time of particles (stratosphere) [s].*

 - char `clim_oh_filename` [LEN]
- Filename of OH climatology.*

 - char `clim_h2o2_filename` [LEN]
- Filename of H2O2 climatology.*

 - int `oh_chem_reaction`
- Reaction type for OH chemistry (0=none, 2=bimolecular, 3=termolecular).*

 - double `oh_chem` [4]
- Coefficients for OH reaction rate (A, E/R or k0, n, kinf, m).*

 - double `oh_chem_beta`
- Beta parameter for diurnal variability of OH.*

 - double `h2o2_chem_cc`
- Cloud cover parameter for H2O2 chemistry.*

 - int `h2o2_chem_reaction`
- Reaction type for H2O2 chemistry (0=none, 1=SO2).*

 - int `chemgrid_nz`
- Number of altitudes of chemistry grid.*

 - double `chemgrid_z0`
- Lower altitude of chemistry grid [km].*

 - double `chemgrid_z1`
- Upper altitude of chemistry grid [km].*

 - int `chemgrid_nx`
- Number of longitudes of chemistry grid.*

 - double `chemgrid_lon0`
- Lower longitude of chemistry grid [deg].*

 - double `chemgrid_lon1`
- Upper longitude of chemistry grid [deg].*

 - int `chemgrid_ny`
- Number of latitudes of chemistry grid.*

 - double `chemgrid_lat0`
- Lower latitude of chemistry grid [deg].*

 - double `chemgrid_lat1`
- Upper latitude of chemistry grid [deg].*

 - double `dry_depo_dp`
- Dry deposition surface layer [hPa].*

 - double `dry_depo_vdep`
- Dry deposition velocity [m/s].*

 - double `wet_depo_pre` [2]
- Coefficients for precipitation calculation.*

 - double `wet_depo_bc_a`
- Coefficient A for wet deposition below cloud (exponential form).*

 - double `wet_depo_bc_b`
- Coefficient B for wet deposition below cloud (exponential form).*

 - double `wet_depo_ic_a`
- Coefficient A for wet deposition in cloud (exponential form).*

 - double `wet_depo_ic_b`
- Coefficient B for wet deposition in cloud (exponential form).*

 - double `wet_depo_ic_h` [3]
- Coefficients for wet deposition in cloud (Henry's law: Hb, Cb, pH).*

- double `wet_depo_bc_h` [2]
Coefficients for wet deposition below cloud (Henry's law: H_b , C_b).
- double `wet_depo_ic_ret_ratio`
Coefficients for wet deposition in cloud: retention ratio.
- double `wet_depo_bc_ret_ratio`
Coefficients for wet deposition below cloud: retention ratio.
- double `psc_h2o`
H₂O volume mixing ratio for PSC analysis.
- double `psc_hno3`
HNO₃ volume mixing ratio for PSC analysis.
- char `atm_basename` [LEN]
Basename of atmospheric data files.
- char `atm_gpfile` [LEN]
Gnuplot file for atmospheric data.
- double `atm_dt_out`
Time step for atmospheric data output [s].
- int `atm_filter`
Time filter for atmospheric data output (0=none, 1=missval, 2=remove).
- int `atm_stride`
Particle index stride for atmospheric data files.
- int `atm_type`
Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF, 3=CLaMS).
- char `csi_basename` [LEN]
Basename of CSI data files.
- double `csi_dt_out`
Time step for CSI data output [s].
- char `csi_obsfile` [LEN]
Observation data file for CSI analysis.
- double `csi_obsmin`
Minimum observation index to trigger detection.
- double `csi_modmin`
Minimum column density to trigger detection [kg/m^2].
- int `csi_nz`
Number of altitudes of gridded CSI data.
- double `csi_z0`
Lower altitude of gridded CSI data [km].
- double `csi_z1`
Upper altitude of gridded CSI data [km].
- int `csi_nx`
Number of longitudes of gridded CSI data.
- double `csi_lon0`
Lower longitude of gridded CSI data [deg].
- double `csi_lon1`
Upper longitude of gridded CSI data [deg].
- int `csi_ny`
Number of latitudes of gridded CSI data.
- double `csi_lat0`
Lower latitude of gridded CSI data [deg].
- double `csi_lat1`
Upper latitude of gridded CSI data [deg].
- char `ens_basename` [LEN]

- *Basename of ensemble data file.*
- double `ens_dt_out`
 - Time step for ensemble output [s].*
- char `grid_basename` [LEN]
 - Basename of grid data files.*
- char `grid_gpfile` [LEN]
 - Gnuplot file for gridded data.*
- double `grid_dt_out`
 - Time step for gridded data output [s].*
- int `grid_sparse`
 - Sparse output in grid data files (0=no, 1=yes).*
- int `grid_nz`
 - Number of altitudes of gridded data.*
- double `grid_z0`
 - Lower altitude of gridded data [km].*
- double `grid_z1`
 - Upper altitude of gridded data [km].*
- int `grid_nx`
 - Number of longitudes of gridded data.*
- double `grid_lon0`
 - Lower longitude of gridded data [deg].*
- double `grid_lon1`
 - Upper longitude of gridded data [deg].*
- int `grid_ny`
 - Number of latitudes of gridded data.*
- double `grid_lat0`
 - Lower latitude of gridded data [deg].*
- double `grid_lat1`
 - Upper latitude of gridded data [deg].*
- int `grid_type`
 - Type of grid data files (0=ASCII, 1=netCDF).*
- char `prof_basename` [LEN]
 - Basename for profile output file.*
- char `prof_obsfile` [LEN]
 - Observation data file for profile output.*
- int `prof_nz`
 - Number of altitudes of gridded profile data.*
- double `prof_z0`
 - Lower altitude of gridded profile data [km].*
- double `prof_z1`
 - Upper altitude of gridded profile data [km].*
- int `prof_nx`
 - Number of longitudes of gridded profile data.*
- double `prof_lon0`
 - Lower longitude of gridded profile data [deg].*
- double `prof_lon1`
 - Upper longitude of gridded profile data [deg].*
- int `prof_ny`
 - Number of latitudes of gridded profile data.*
- double `prof_lat0`
 - Lower latitude of gridded profile data [deg].*

- double `prof_lat1`
Upper latitude of gridded profile data [deg].
- char `sample_basename` [LEN]
Basename of sample data file.
- char `sample_obsfile` [LEN]
Observation data file for sample output.
- double `sample_dx`
Horizontal radius for sample output [km].
- double `sample_dz`
Layer depth for sample output [km].
- char `stat_basename` [LEN]
Basename of station data file.
- double `stat_lon`
Longitude of station [deg].
- double `stat_lat`
Latitude of station [deg].
- double `stat_r`
Search radius around station [km].
- double `stat_t0`
Start time for station output [s].
- double `stat_t1`
Stop time for station output [s].

4.4.1 Detailed Description

Control parameters.

Definition at line 697 of file `libtrac.h`.

4.4.2 Field Documentation

4.4.2.1 `vert_coord_ap` `int` `ctl_t::vert_coord_ap`

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

Definition at line 700 of file `libtrac.h`.

4.4.2.2 `vert_coord_met` `int` `ctl_t::vert_coord_met`

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

Definition at line 703 of file `libtrac.h`.

4.4.2.3 vert_vel `int ctl_t::vert_vel`

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

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

4.4.2.4 clams_met_data `int ctl_t::clams_met_data`

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

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

4.4.2.5 chunkszhint `size_t ctl_t::chunkszhint`

Chunk size hint for nc__open.

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

4.4.2.6 read_mode `int ctl_t::read_mode`

Read mode for nc__open.

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

4.4.2.7 nq `int ctl_t::nq`

Number of quantities.

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

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

Quantity names.

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

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

Quantity long names.

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

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

Quantity units.

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

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

Quantity output format.

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

4.4.2.12 `qnt_idx` `int ctl_t::qnt_idx`

Quantity array index for air parcel IDs.

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

4.4.2.13 `qnt_ens` `int ctl_t::qnt_ens`

Quantity array index for ensemble IDs.

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

4.4.2.14 `qnt_stat` `int ctl_t::qnt_stat`

Quantity array index for station flag.

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

4.4.2.15 `qnt_m` `int ctl_t::qnt_m`

Quantity array index for mass.

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

4.4.2.16 `qnt_vmr` `int ctl_t::qnt_vmr`

Quantity array index for volume mixing ratio.

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

4.4.2.17 `qnt_rp` `int ctl_t::qnt_rp`

Quantity array index for particle radius.

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

4.4.2.18 `qnt_rhop` `int ctl_t::qnt_rhop`

Quantity array index for particle density.

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

4.4.2.19 `qnt_ps` `int ctl_t::qnt_ps`

Quantity array index for surface pressure.

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

4.4.2.20 `qnt_ts` `int ctl_t::qnt_ts`

Quantity array index for surface temperature.

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

4.4.2.21 `qnt_zs` `int` `ctl_t::qnt_zs`

Quantity array index for surface geopotential height.

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

4.4.2.22 `qnt_us` `int` `ctl_t::qnt_us`

Quantity array index for surface zonal wind.

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

4.4.2.23 `qnt_vs` `int` `ctl_t::qnt_vs`

Quantity array index for surface meridional wind.

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

4.4.2.24 `qnt_pbl` `int` `ctl_t::qnt_pbl`

Quantity array index for boundary layer pressure.

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

4.4.2.25 `qnt_pt` `int` `ctl_t::qnt_pt`

Quantity array index for tropopause pressure.

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

4.4.2.26 `qnt_tt` `int` `ctl_t::qnt_tt`

Quantity array index for tropopause temperature.

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

4.4.2.27 qnt_zt `int ctl_t::qnt_zt`

Quantity array index for tropopause geopotential height.

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

4.4.2.28 qnt_h2ot `int ctl_t::qnt_h2ot`

Quantity array index for tropopause water vapor vmr.

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

4.4.2.29 qnt_z `int ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line [784](#) of file [libtrac.h](#).

4.4.2.30 qnt_p `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line [787](#) of file [libtrac.h](#).

4.4.2.31 qnt_t `int ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line [790](#) of file [libtrac.h](#).

4.4.2.32 qnt_rho `int ctl_t::qnt_rho`

Quantity array index for density of air.

Definition at line [793](#) of file [libtrac.h](#).

4.4.2.33 `qnt_u` `int` `ctl_t::qnt_u`

Quantity array index for zonal wind.

Definition at line 796 of file [libtrac.h](#).

4.4.2.34 `qnt_v` `int` `ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line 799 of file [libtrac.h](#).

4.4.2.35 `qnt_w` `int` `ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line 802 of file [libtrac.h](#).

4.4.2.36 `qnt_h2o` `int` `ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line 805 of file [libtrac.h](#).

4.4.2.37 `qnt_o3` `int` `ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line 808 of file [libtrac.h](#).

4.4.2.38 `qnt_lwc` `int` `ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line 811 of file [libtrac.h](#).

4.4.2.39 qnt_iwc `int ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line [814](#) of file [libtrac.h](#).

4.4.2.40 qnt_pct `int ctl_t::qnt_pct`

Quantity array index for cloud top pressure.

Definition at line [817](#) of file [libtrac.h](#).

4.4.2.41 qnt_pcb `int ctl_t::qnt_pcb`

Quantity array index for cloud bottom pressure.

Definition at line [820](#) of file [libtrac.h](#).

4.4.2.42 qnt_cl `int ctl_t::qnt_cl`

Quantity array index for total column cloud water.

Definition at line [823](#) of file [libtrac.h](#).

4.4.2.43 qnt_plcl `int ctl_t::qnt_plcl`

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

Definition at line [826](#) of file [libtrac.h](#).

4.4.2.44 qnt_plfc `int ctl_t::qnt_plfc`

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

Definition at line [829](#) of file [libtrac.h](#).

4.4.2.45 `qnt_pel` `int` `ctl_t::qnt_pel`

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

Definition at line 832 of file [libtrac.h](#).

4.4.2.46 `qnt_cape` `int` `ctl_t::qnt_cape`

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

Definition at line 835 of file [libtrac.h](#).

4.4.2.47 `qnt_cin` `int` `ctl_t::qnt_cin`

Quantity array index for convective inhibition (CIN).

Definition at line 838 of file [libtrac.h](#).

4.4.2.48 `qnt_hno3` `int` `ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line 841 of file [libtrac.h](#).

4.4.2.49 `qnt_oh` `int` `ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line 844 of file [libtrac.h](#).

4.4.2.50 `qnt_vmrimpl` `int` `ctl_t::qnt_vmrimpl`

Quantity array index for implicity volumn mixing ratio.

Definition at line 847 of file [libtrac.h](#).

4.4.2.51 qnt_mloss_oh `int ctl_t::qnt_mloss_oh`

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

Definition at line [850](#) of file [libtrac.h](#).

4.4.2.52 qnt_mloss_h2o2 `int ctl_t::qnt_mloss_h2o2`

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

Definition at line [853](#) of file [libtrac.h](#).

4.4.2.53 qnt_mloss_wet `int ctl_t::qnt_mloss_wet`

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

Definition at line [856](#) of file [libtrac.h](#).

4.4.2.54 qnt_mloss_dry `int ctl_t::qnt_mloss_dry`

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

Definition at line [859](#) of file [libtrac.h](#).

4.4.2.55 qnt_mloss_decay `int ctl_t::qnt_mloss_decay`

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

Definition at line [862](#) of file [libtrac.h](#).

4.4.2.56 qnt_psat `int ctl_t::qnt_psat`

Quantity array index for saturation pressure over water.

Definition at line [865](#) of file [libtrac.h](#).

4.4.2.57 `qnt_psice` `int` `ctl_t::qnt_psice`

Quantity array index for saturation pressure over ice.

Definition at line [868](#) of file [libtrac.h](#).

4.4.2.58 `qnt_pw` `int` `ctl_t::qnt_pw`

Quantity array index for partial water vapor pressure.

Definition at line [871](#) of file [libtrac.h](#).

4.4.2.59 `qnt_sh` `int` `ctl_t::qnt_sh`

Quantity array index for specific humidity.

Definition at line [874](#) of file [libtrac.h](#).

4.4.2.60 `qnt_rh` `int` `ctl_t::qnt_rh`

Quantity array index for relative humidity over water.

Definition at line [877](#) of file [libtrac.h](#).

4.4.2.61 `qnt_rhice` `int` `ctl_t::qnt_rhice`

Quantity array index for relative humidity over ice.

Definition at line [880](#) of file [libtrac.h](#).

4.4.2.62 `qnt_theta` `int` `ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line [883](#) of file [libtrac.h](#).

4.4.2.63 qnt_zeta `int ctl_t::qnt_zeta`

Quantity array index for zeta vertical coordinate.

Definition at line [886](#) of file [libtrac.h](#).

4.4.2.64 qnt_tvirt `int ctl_t::qnt_tvirt`

Quantity array index for virtual temperature.

Definition at line [889](#) of file [libtrac.h](#).

4.4.2.65 qnt_lapse `int ctl_t::qnt_lapse`

Quantity array index for lapse rate.

Definition at line [892](#) of file [libtrac.h](#).

4.4.2.66 qnt_vh `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line [895](#) of file [libtrac.h](#).

4.4.2.67 qnt_vz `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line [898](#) of file [libtrac.h](#).

4.4.2.68 qnt_pv `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line [901](#) of file [libtrac.h](#).

4.4.2.69 `qnt_tdew` `int` `ctl_t::qnt_tdew`

Quantity array index for dew point temperature.

Definition at line 904 of file [libtrac.h](#).

4.4.2.70 `qnt_tice` `int` `ctl_t::qnt_tice`

Quantity array index for T_ice.

Definition at line 907 of file [libtrac.h](#).

4.4.2.71 `qnt_tsts` `int` `ctl_t::qnt_tsts`

Quantity array index for T_STS.

Definition at line 910 of file [libtrac.h](#).

4.4.2.72 `qnt_tnat` `int` `ctl_t::qnt_tnat`

Quantity array index for T_NAT.

Definition at line 913 of file [libtrac.h](#).

4.4.2.73 `direction` `int` `ctl_t::direction`

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

Definition at line 916 of file [libtrac.h](#).

4.4.2.74 `t_start` `double` `ctl_t::t_start`

Start time of simulation [s].

Definition at line 919 of file [libtrac.h](#).

4.4.2.75 t_stop `double ctl_t::t_stop`

Stop time of simulation [s].

Definition at line [922](#) of file [libtrac.h](#).

4.4.2.76 dt_mod `double ctl_t::dt_mod`

Time step of simulation [s].

Definition at line [925](#) of file [libtrac.h](#).

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

Basename for meteo data.

Definition at line [928](#) of file [libtrac.h](#).

4.4.2.78 dt_met `double ctl_t::dt_met`

Time step of meteo data [s].

Definition at line [931](#) of file [libtrac.h](#).

4.4.2.79 met_type `int ctl_t::met_type`

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

Definition at line [934](#) of file [libtrac.h](#).

4.4.2.80 met_nc_scale `int ctl_t::met_nc_scale`

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

Definition at line [937](#) of file [libtrac.h](#).

4.4.2.81 `met_dx` `int` `ctl_t::met_dx`

Stride for longitudes.

Definition at line 940 of file [libtrac.h](#).

4.4.2.82 `met_dy` `int` `ctl_t::met_dy`

Stride for latitudes.

Definition at line 943 of file [libtrac.h](#).

4.4.2.83 `met_dp` `int` `ctl_t::met_dp`

Stride for pressure levels.

Definition at line 946 of file [libtrac.h](#).

4.4.2.84 `met_sx` `int` `ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 949 of file [libtrac.h](#).

4.4.2.85 `met_sy` `int` `ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 952 of file [libtrac.h](#).

4.4.2.86 `met_sp` `int` `ctl_t::met_sp`

Smoothing for pressure levels.

Definition at line 955 of file [libtrac.h](#).

4.4.2.87 met_detrend `double ctl_t::met_detrend`

FWHM of horizontal Gaussian used for detrending [km].

Definition at line [958](#) of file [libtrac.h](#).

4.4.2.88 met_np `int ctl_t::met_np`

Number of target pressure levels.

Definition at line [961](#) of file [libtrac.h](#).

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

Target pressure levels [hPa].

Definition at line [964](#) of file [libtrac.h](#).

4.4.2.90 met_geopot_sx `int ctl_t::met_geopot_sx`

Longitudinal smoothing of geopotential heights.

Definition at line [967](#) of file [libtrac.h](#).

4.4.2.91 met_geopot_sy `int ctl_t::met_geopot_sy`

Latitudinal smoothing of geopotential heights.

Definition at line [970](#) of file [libtrac.h](#).

4.4.2.92 met_relhum `int ctl_t::met_relhum`

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

Definition at line [973](#) of file [libtrac.h](#).

4.4.2.93 `met_tropo` `int ctl_t::met_tropo`

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

Definition at line 977 of file [libtrac.h](#).

4.4.2.94 `met_tropo_lapse` `double ctl_t::met_tropo_lapse`

WMO tropopause lapse rate [K/km].

Definition at line 980 of file [libtrac.h](#).

4.4.2.95 `met_tropo_nlev` `int ctl_t::met_tropo_nlev`

WMO tropopause layer depth (number of levels).

Definition at line 983 of file [libtrac.h](#).

4.4.2.96 `met_tropo_lapse_sep` `double ctl_t::met_tropo_lapse_sep`

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

Definition at line 986 of file [libtrac.h](#).

4.4.2.97 `met_tropo_nlev_sep` `int ctl_t::met_tropo_nlev_sep`

WMO tropopause separation layer depth (number of levels).

Definition at line 989 of file [libtrac.h](#).

4.4.2.98 `met_tropo_pv` `double ctl_t::met_tropo_pv`

Dynamical tropopause potential vorticity threshold [PVU].

Definition at line 992 of file [libtrac.h](#).

4.4.2.99 met_tropo_theta `double ctl_t::met_tropo_theta`

Dynamical tropopause potential temperature threshold [K].

Definition at line [995](#) of file [libtrac.h](#).

4.4.2.100 met_tropo_spline `int ctl_t::met_tropo_spline`

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

Definition at line [998](#) of file [libtrac.h](#).

4.4.2.101 met_cloud `int ctl_t::met_cloud`

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

Definition at line [1001](#) of file [libtrac.h](#).

4.4.2.102 met_cloud_min `double ctl_t::met_cloud_min`

Minimum cloud ice water content [kg/kg].

Definition at line [1004](#) of file [libtrac.h](#).

4.4.2.103 met_dt_out `double ctl_t::met_dt_out`

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

Definition at line [1007](#) of file [libtrac.h](#).

4.4.2.104 met_cache `int ctl_t::met_cache`

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

Definition at line [1010](#) of file [libtrac.h](#).

4.4.2.105 `sort_dt` `double ctl_t::sort_dt`

Time step for sorting of particle data [s].

Definition at line 1013 of file [libtrac.h](#).

4.4.2.106 `isosurf` `int ctl_t::isosurf`

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

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

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

Balloon position filename.

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

4.4.2.108 `advect` `int ctl_t::advect`

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

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

4.4.2.109 `reflect` `int ctl_t::reflect`

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

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

4.4.2.110 `turb_dx_trop` `double ctl_t::turb_dx_trop`

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

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

4.4.2.111 turb_dx_strat `double ctl_t::turb_dx_strat`

Horizontal turbulent diffusion coefficient (stratosphere) [m²/s].

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

4.4.2.112 turb_dz_trop `double ctl_t::turb_dz_trop`

Vertical turbulent diffusion coefficient (troposphere) [m²/s].

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

4.4.2.113 turb_dz_strat `double ctl_t::turb_dz_strat`

Vertical turbulent diffusion coefficient (stratosphere) [m²/s].

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

4.4.2.114 turb_mesox `double ctl_t::turb_mesox`

Horizontal scaling factor for mesoscale wind fluctuations.

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

4.4.2.115 turb_mesoz `double ctl_t::turb_mesoz`

Vertical scaling factor for mesoscale wind fluctuations.

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

4.4.2.116 conv_cape `double ctl_t::conv_cape`

CAPE threshold for convection module [J/kg].

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

4.4.2.117 `conv_cin` `double ctl_t::conv_cin`

CIN threshold for convection module [J/kg].

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

4.4.2.118 `conv_dt` `double ctl_t::conv_dt`

Time interval for convection module [s].

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

4.4.2.119 `conv_mix` `int ctl_t::conv_mix`

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

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

4.4.2.120 `conv_mix_bot` `int ctl_t::conv_mix_bot`

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

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

4.4.2.121 `conv_mix_top` `int ctl_t::conv_mix_top`

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

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

4.4.2.122 `bound_mass` `double ctl_t::bound_mass`

Boundary conditions mass per particle [kg].

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

4.4.2.123 bound_mass_trend `double ctl_t::bound_mass_trend`

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

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

4.4.2.124 bound_vmr `double ctl_t::bound_vmr`

Boundary conditions volume mixing ratio [ppv].

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

4.4.2.125 bound_vmr_trend `double ctl_t::bound_vmr_trend`

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

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

4.4.2.126 bound_lat0 `double ctl_t::bound_lat0`

Boundary conditions minimum longitude [deg].

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

4.4.2.127 bound_lat1 `double ctl_t::bound_lat1`

Boundary conditions maximum longitude [deg].

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

4.4.2.128 bound_p0 `double ctl_t::bound_p0`

Boundary conditions bottom pressure [hPa].

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

4.4.2.129 `bound_p1` `double ctl_t::bound_p1`

Boundary conditions top pressure [hPa].

Definition at line [1086](#) of file [libtrac.h](#).

4.4.2.130 `bound_dps` `double ctl_t::bound_dps`

Boundary conditions surface layer depth [hPa].

Definition at line [1089](#) of file [libtrac.h](#).

4.4.2.131 `bound_dzs` `double ctl_t::bound_dzs`

Boundary conditions surface layer depth [km].

Definition at line [1092](#) of file [libtrac.h](#).

4.4.2.132 `bound_zetas` `double ctl_t::bound_zetas`

Boundary conditions surface layer zeta [K].

Definition at line [1095](#) of file [libtrac.h](#).

4.4.2.133 `bound_pbl` `int ctl_t::bound_pbl`

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

Definition at line [1098](#) of file [libtrac.h](#).

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

Species.

Definition at line [1101](#) of file [libtrac.h](#).

4.4.2.135 molmass `double ctl_t::molmass`

Molar mass [g/mol].

Definition at line [1104](#) of file [libtrac.h](#).

4.4.2.136 tdec_trop `double ctl_t::tdec_trop`

Life time of particles (troposphere) [s].

Definition at line [1107](#) of file [libtrac.h](#).

4.4.2.137 tdec_strat `double ctl_t::tdec_strat`

Life time of particles (stratosphere) [s].

Definition at line [1110](#) of file [libtrac.h](#).

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

Filename of OH climatology.

Definition at line [1113](#) of file [libtrac.h](#).

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

Filename of H2O2 climatology.

Definition at line [1116](#) of file [libtrac.h](#).

4.4.2.140 oh_chem_reaction `int ctl_t::oh_chem_reaction`

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

Definition at line [1119](#) of file [libtrac.h](#).

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

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

Definition at line 1122 of file [libtrac.h](#).

4.4.2.142 `oh_chem_beta` `double ctl_t::oh_chem_beta`

Beta parameter for diurnal variability of OH.

Definition at line 1125 of file [libtrac.h](#).

4.4.2.143 `h2o2_chem_cc` `double ctl_t::h2o2_chem_cc`

Cloud cover parameter for H2O2 chemistry.

Definition at line 1128 of file [libtrac.h](#).

4.4.2.144 `h2o2_chem_reaction` `int ctl_t::h2o2_chem_reaction`

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

Definition at line 1131 of file [libtrac.h](#).

4.4.2.145 `chemgrid_nz` `int ctl_t::chemgrid_nz`

Number of altitudes of chemistry grid.

Definition at line 1134 of file [libtrac.h](#).

4.4.2.146 `chemgrid_z0` `double ctl_t::chemgrid_z0`

Lower altitude of chemistry grid [km].

Definition at line 1137 of file [libtrac.h](#).

4.4.2.147 chemgrid_z1 `double ctl_t::chemgrid_z1`

Upper altitude of chemistry grid [km].

Definition at line [1140](#) of file [libtrac.h](#).

4.4.2.148 chemgrid_nx `int ctl_t::chemgrid_nx`

Number of longitudes of chemistry grid.

Definition at line [1143](#) of file [libtrac.h](#).

4.4.2.149 chemgrid_lon0 `double ctl_t::chemgrid_lon0`

Lower longitude of chemistry grid [deg].

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

4.4.2.150 chemgrid_lon1 `double ctl_t::chemgrid_lon1`

Upper longitude of chemistry grid [deg].

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

4.4.2.151 chemgrid_ny `int ctl_t::chemgrid_ny`

Number of latitudes of chemistry grid.

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

4.4.2.152 chemgrid_lat0 `double ctl_t::chemgrid_lat0`

Lower latitude of chemistry grid [deg].

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

4.4.2.153 `chemgrid_lat1` `double ctl_t::chemgrid_lat1`

Upper latitude of chemistry grid [deg].

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

4.4.2.154 `dry_depo_dp` `double ctl_t::dry_depo_dp`

Dry deposition surface layer [hPa].

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

4.4.2.155 `dry_depo_vdep` `double ctl_t::dry_depo_vdep`

Dry deposition velocity [m/s].

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

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

Coefficients for precipitation calculation.

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

4.4.2.157 `wet_depo_bc_a` `double ctl_t::wet_depo_bc_a`

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

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

4.4.2.158 `wet_depo_bc_b` `double ctl_t::wet_depo_bc_b`

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

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

4.4.2.159 wet_depo_ic_a `double ctl_t::wet_depo_ic_a`

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

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

4.4.2.160 wet_depo_ic_b `double ctl_t::wet_depo_ic_b`

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

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

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

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

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

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

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

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

4.4.2.163 wet_depo_ic_ret_ratio `double ctl_t::wet_depo_ic_ret_ratio`

Coefficients for wet deposition in cloud: retention ratio.

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

4.4.2.164 wet_depo_bc_ret_ratio `double ctl_t::wet_depo_bc_ret_ratio`

Coefficients for wet deposition below cloud: retention ratio.

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

4.4.2.165 `psc_h2o` `double ctl_t::psc_h2o`

H2O volume mixing ratio for PSC analysis.

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

4.4.2.166 `psc_hno3` `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

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

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

Basename of atmospheric data files.

Definition at line 1200 of file [libtrac.h](#).

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

Gnuplot file for atmospheric data.

Definition at line 1203 of file [libtrac.h](#).

4.4.2.169 `atm_dt_out` `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

Definition at line 1206 of file [libtrac.h](#).

4.4.2.170 `atm_filter` `int ctl_t::atm_filter`

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

Definition at line 1209 of file [libtrac.h](#).

4.4.2.171 atm_stride `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line [1212](#) of file [libtrac.h](#).

4.4.2.172 atm_type `int ctl_t::atm_type`

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

Definition at line [1215](#) of file [libtrac.h](#).

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

Basename of CSI data files.

Definition at line [1218](#) of file [libtrac.h](#).

4.4.2.174 csi_dt_out `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line [1221](#) of file [libtrac.h](#).

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

Observation data file for CSI analysis.

Definition at line [1224](#) of file [libtrac.h](#).

4.4.2.176 csi_obsmin `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line [1227](#) of file [libtrac.h](#).

4.4.2.177 `csi_modmin` `double ctl_t::csi_modmin`

Minimum column density to trigger detection [kg/m²].

Definition at line 1230 of file [libtrac.h](#).

4.4.2.178 `csi_nz` `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line 1233 of file [libtrac.h](#).

4.4.2.179 `csi_z0` `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

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

4.4.2.180 `csi_z1` `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

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

4.4.2.181 `csi_nx` `int ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

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

4.4.2.182 `csi_lon0` `double ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 1245 of file [libtrac.h](#).

4.4.2.183 `csi_lon1` `double` `ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line [1248](#) of file [libtrac.h](#).

4.4.2.184 `csi_ny` `int` `ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line [1251](#) of file [libtrac.h](#).

4.4.2.185 `csi_lat0` `double` `ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line [1254](#) of file [libtrac.h](#).

4.4.2.186 `csi_lat1` `double` `ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

Definition at line [1257](#) of file [libtrac.h](#).

4.4.2.187 `ens_basename` `char` `ctl_t::ens_basename`[[LEN](#)]

Basename of ensemble data file.

Definition at line [1260](#) of file [libtrac.h](#).

4.4.2.188 `ens_dt_out` `double` `ctl_t::ens_dt_out`

Time step for ensemble output [s].

Definition at line [1263](#) of file [libtrac.h](#).

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

Basename of grid data files.

Definition at line 1266 of file [libtrac.h](#).

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

Gnuplot file for gridded data.

Definition at line 1269 of file [libtrac.h](#).

4.4.2.191 `grid_dt_out` `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

Definition at line 1272 of file [libtrac.h](#).

4.4.2.192 `grid_sparse` `int ctl_t::grid_sparse`

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

Definition at line 1275 of file [libtrac.h](#).

4.4.2.193 `grid_nz` `int ctl_t::grid_nz`

Number of altitudes of gridded data.

Definition at line 1278 of file [libtrac.h](#).

4.4.2.194 `grid_z0` `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

Definition at line 1281 of file [libtrac.h](#).

4.4.2.195 grid_z1 `double ctl_t::grid_z1`

Upper altitude of gridded data [km].

Definition at line [1284](#) of file [libtrac.h](#).

4.4.2.196 grid_nx `int ctl_t::grid_nx`

Number of longitudes of gridded data.

Definition at line [1287](#) of file [libtrac.h](#).

4.4.2.197 grid_lon0 `double ctl_t::grid_lon0`

Lower longitude of gridded data [deg].

Definition at line [1290](#) of file [libtrac.h](#).

4.4.2.198 grid_lon1 `double ctl_t::grid_lon1`

Upper longitude of gridded data [deg].

Definition at line [1293](#) of file [libtrac.h](#).

4.4.2.199 grid_ny `int ctl_t::grid_ny`

Number of latitudes of gridded data.

Definition at line [1296](#) of file [libtrac.h](#).

4.4.2.200 grid_lat0 `double ctl_t::grid_lat0`

Lower latitude of gridded data [deg].

Definition at line [1299](#) of file [libtrac.h](#).

4.4.2.201 `grid_lat1` `double ctl_t::grid_lat1`

Upper latitude of gridded data [deg].

Definition at line 1302 of file [libtrac.h](#).

4.4.2.202 `grid_type` `int ctl_t::grid_type`

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

Definition at line 1305 of file [libtrac.h](#).

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

Basename for profile output file.

Definition at line 1308 of file [libtrac.h](#).

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

Observation data file for profile output.

Definition at line 1311 of file [libtrac.h](#).

4.4.2.205 `prof_nz` `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

Definition at line 1314 of file [libtrac.h](#).

4.4.2.206 `prof_z0` `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

Definition at line 1317 of file [libtrac.h](#).

4.4.2.207 **prof_z1** `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

Definition at line [1320](#) of file [libtrac.h](#).

4.4.2.208 **prof_nx** `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

Definition at line [1323](#) of file [libtrac.h](#).

4.4.2.209 **prof_lon0** `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

Definition at line [1326](#) of file [libtrac.h](#).

4.4.2.210 **prof_lon1** `double ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

Definition at line [1329](#) of file [libtrac.h](#).

4.4.2.211 **prof_ny** `int ctl_t::prof_ny`

Number of latitudes of gridded profile data.

Definition at line [1332](#) of file [libtrac.h](#).

4.4.2.212 **prof_lat0** `double ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

Definition at line [1335](#) of file [libtrac.h](#).

4.4.2.213 `prof_lat1` `double ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

Definition at line 1338 of file [libtrac.h](#).

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

Basename of sample data file.

Definition at line 1341 of file [libtrac.h](#).

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

Observation data file for sample output.

Definition at line 1344 of file [libtrac.h](#).

4.4.2.216 `sample_dx` `double ctl_t::sample_dx`

Horizontal radius for sample output [km].

Definition at line 1347 of file [libtrac.h](#).

4.4.2.217 `sample_dz` `double ctl_t::sample_dz`

Layer depth for sample output [km].

Definition at line 1350 of file [libtrac.h](#).

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

Basename of station data file.

Definition at line 1353 of file [libtrac.h](#).

4.4.2.219 stat_lon `double ctl_t::stat_lon`

Longitude of station [deg].

Definition at line [1356](#) of file [libtrac.h](#).

4.4.2.220 stat_lat `double ctl_t::stat_lat`

Latitude of station [deg].

Definition at line [1359](#) of file [libtrac.h](#).

4.4.2.221 stat_r `double ctl_t::stat_r`

Search radius around station [km].

Definition at line [1362](#) of file [libtrac.h](#).

4.4.2.222 stat_t0 `double ctl_t::stat_t0`

Start time for station output [s].

Definition at line [1365](#) of file [libtrac.h](#).

4.4.2.223 stat_t1 `double ctl_t::stat_t1`

Stop time for station output [s].

Definition at line [1368](#) of file [libtrac.h](#).

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

- [libtrac.h](#)

4.5 met_t Struct Reference

Meteo data.

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

Data Fields

- double [time](#)
Time [s].
- int [nx](#)
Number of longitudes.
- int [ny](#)
Number of latitudes.
- int [np](#)
Number of pressure levels.
- double [lon](#) [[EX](#)]
Longitude [deg].
- double [lat](#) [[EY](#)]
Latitude [deg].
- double [p](#) [[EP](#)]
Pressure [hPa].
- float [ps](#) [[EX](#)][[EY](#)]
Surface pressure [hPa].
- float [ts](#) [[EX](#)][[EY](#)]
Surface temperature [K].
- float [zs](#) [[EX](#)][[EY](#)]
Surface geopotential height [km].
- float [us](#) [[EX](#)][[EY](#)]
Surface zonal wind [m/s].
- float [vs](#) [[EX](#)][[EY](#)]
Surface meridional wind [m/s].
- float [pbl](#) [[EX](#)][[EY](#)]
Boundary layer pressure [hPa].
- float [pt](#) [[EX](#)][[EY](#)]
Tropopause pressure [hPa].
- float [tt](#) [[EX](#)][[EY](#)]
Tropopause temperature [K].
- float [zt](#) [[EX](#)][[EY](#)]
Tropopause geopotential height [km].
- float [h2ot](#) [[EX](#)][[EY](#)]
Tropopause water vapor vmr [ppv].
- float [pct](#) [[EX](#)][[EY](#)]
Cloud top pressure [hPa].
- float [pcb](#) [[EX](#)][[EY](#)]
Cloud bottom pressure [hPa].
- float [cl](#) [[EX](#)][[EY](#)]
Total column cloud water [kg/m^2].
- float [plcl](#) [[EX](#)][[EY](#)]
Pressure at lifted condensation level (LCL) [hPa].
- float [plfc](#) [[EX](#)][[EY](#)]
Pressure at level of free convection (LFC) [hPa].
- float [pel](#) [[EX](#)][[EY](#)]
Pressure at equilibrium level [hPa].
- float [cape](#) [[EX](#)][[EY](#)]
Convective available potential energy [J/kg].
- float [cin](#) [[EX](#)][[EY](#)]

- float `z` [EX][EY][EP]
Convective inhibition [J/kg].
- float `t` [EX][EY][EP]
Geopotential height [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 `patp` [EX][EY][EP]
Pressure on model levels [hPa].
- float `zeta` [EX][EY][EP]
Pressure field in pressure levels [hPa].
- float `zeta_dot` [EX][EY][EP]
Zeta [K].
- float `zeta_dot` [EX][EY][EP]
Vertical velocity [K/s].

4.5.1 Detailed Description

Meteo data.

Definition at line 1502 of file `libtrac.h`.

4.5.2 Field Documentation

4.5.2.1 `time` double met_t::time

Time [s].

Definition at line 1505 of file `libtrac.h`.

4.5.2.2 nx `int met_t::nx`

Number of longitudes.

Definition at line [1508](#) of file [libtrac.h](#).

4.5.2.3 ny `int met_t::ny`

Number of latitudes.

Definition at line [1511](#) of file [libtrac.h](#).

4.5.2.4 np `int met_t::np`

Number of pressure levels.

Definition at line [1514](#) of file [libtrac.h](#).

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

Longitude [deg].

Definition at line [1517](#) of file [libtrac.h](#).

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

Latitude [deg].

Definition at line [1520](#) of file [libtrac.h](#).

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

Pressure [hPa].

Definition at line [1523](#) of file [libtrac.h](#).

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

Surface pressure [hPa].

Definition at line 1526 of file libtrac.h.

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

Surface temperature [K].

Definition at line 1529 of file libtrac.h.

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

Surface geopotential height [km].

Definition at line 1532 of file libtrac.h.

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

Surface zonal wind [m/s].

Definition at line 1535 of file libtrac.h.

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

Surface meridional wind [m/s].

Definition at line 1538 of file libtrac.h.

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

Boundary layer pressure [hPa].

Definition at line 1541 of file libtrac.h.

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

Tropopause pressure [hPa].

Definition at line 1544 of file libtrac.h.

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

Tropopause temperature [K].

Definition at line 1547 of file libtrac.h.

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

Tropopause geopotential height [km].

Definition at line 1550 of file libtrac.h.

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

Tropopause water vapor vmr [ppv].

Definition at line 1553 of file libtrac.h.

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

Cloud top pressure [hPa].

Definition at line 1556 of file libtrac.h.

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

Cloud bottom pressure [hPa].

Definition at line 1559 of file libtrac.h.

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

Total column cloud water [kg/m²].

Definition at line 1562 of file libtrac.h.

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

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

Definition at line 1565 of file libtrac.h.

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

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

Definition at line 1568 of file libtrac.h.

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

Pressure at equilibrium level [hPa].

Definition at line 1571 of file libtrac.h.

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

Convective available potential energy [J/kg].

Definition at line 1574 of file libtrac.h.

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

Convective inhibition [J/kg].

Definition at line 1577 of file libtrac.h.

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

Geopotential height [km].

Definition at line 1580 of file libtrac.h.

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

Temperature [K].

Definition at line 1583 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 1586 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 1589 of file libtrac.h.

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

Vertical velocity [hPa/s].

Definition at line 1592 of file libtrac.h.

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

Potential vorticity [PVU].

Definition at line 1595 of file libtrac.h.

4.5.2.32 h2o `float met_t::h2o`[\[EX\]](#) [\[EY\]](#) [\[EP\]](#)

Water vapor volume mixing ratio [1].

Definition at line [1598](#) of file [libtrac.h](#).

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

Ozone volume mixing ratio [1].

Definition at line [1601](#) of file [libtrac.h](#).

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

Cloud liquid water content [kg/kg].

Definition at line [1604](#) of file [libtrac.h](#).

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

Cloud ice water content [kg/kg].

Definition at line [1607](#) of file [libtrac.h](#).

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

Pressure on model levels [hPa].

Definition at line [1610](#) of file [libtrac.h](#).

4.5.2.37 patp `float met_t::patp`[\[EX\]](#) [\[EY\]](#) [\[EP\]](#)

Pressure field in pressure levels [hPa].

Definition at line [1613](#) of file [libtrac.h](#).

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

Zeta [K].

Definition at line 1616 of file libtrac.h.

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

Vertical velocity [K/s].

Definition at line 1619 of file libtrac.h.

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

- libtrac.h

5 File Documentation

5.1 atm_conv.c File Reference

Convert file format of air parcel data files.

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

Functions

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

5.1.1 Detailed Description

Convert file format of air parcel data files.

Definition in file atm_conv.c.

5.1.2 Function Documentation

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

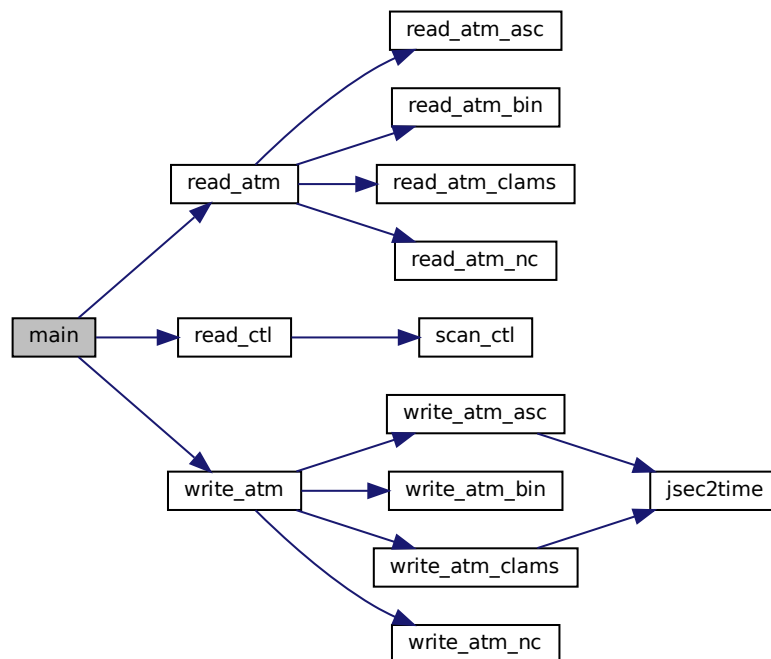
Definition at line 27 of file atm_conv.c.

```

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) 2013–2022 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  int main(
00028      int argc,
00029      char *argv[]) {
00030
00031      ctl_t ctl;
00032
00033      atm_t *atm;
00034
00035      /* Check arguments... */
00036      if (argc < 6)
00037          ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038                " <atm_out> <atm_out_type>");
00039
00040      /* Allocate... */
00041      ALLOC(atm, atm_t, 1);
00042
00043      /* Read control parameters... */
00044      read_ctl(argv[1], argc, argv, &ctl);
00045
00046      /* Read atmospheric data... */
00047      ctl.atm_type = atoi(argv[3]);
00048      if (!read_atm(argv[2], &ctl, atm))
00049          ERRMSG("Cannot open file!");
00050
00051      /* Write atmospheric data... */
00052      ctl.atm_type = atoi(argv[5]);
00053      write_atm(argv[4], &ctl, atm, 0);
00054
00055      /* Free... */
00056      free(atm);
00057
00058      return EXIT_SUCCESS;
00059 }

```

5.3 atm_dist.c File Reference

Calculate transport deviations of trajectories.

```
#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> <atm1a> <atm1b>"
00087             " [<atm2a> <atm2b> ...]");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092     p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093     p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094     lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095     lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096     lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097     lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098     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         "# $qd = %s absolute difference (%s) [%s]\n"
00119         "# $qd = %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, "# $qd = 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     size_t len = strlen(argv[f]);
00137     sprintf(tstr, "%.4s", &argv[f][len - 20]);
00138     year = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][len - 15]);
00140     mon = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][len - 12]);
00142     day = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][len - 9]);
00144     hour = atoi(tstr);
00145     sprintf(tstr, "%.2s", &argv[f][len - 6]);
00146     min = atoi(tstr);
00147     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00148
00149     /* Check time... */
00150     if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00151         || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00152         ERRMSG("Cannot read time from filename!");
00153
00154     /* Save initial time... */
00155     if (!init) {
00156         init = 1;
00157         t0 = t;
00158     }
00159
00160     /* Init... */
00161     np = 0;
00162     for (ip = 0; ip < atm1->np; ip++) {
00163         ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00164         for (iq = 0; iq < ctl.nq; iq++)
00165             aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00166     }
00167
00168     /* Loop over air parcels... */
00169     for (ip = 0; ip < atm1->np; ip++) {
00170
00171         /* Check air parcel index... */
00172         if (ctl.qnt_idx > 0
00173             && (atm1->q[ctl.qnt_idx][ip] != atm2->q[ctl.qnt_idx][ip]))
00174             ERRMSG("Air parcel index does not match!");
00175
00176         /* Check ensemble index... */
00177         if (ctl.qnt_ens > 0
00178             && (atm1->q[ctl.qnt_ens][ip] != ens
00179                 || atm2->q[ctl.qnt_ens][ip] != ens))
00180             continue;
00181
00182         /* Check time... */
00183         if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00184             continue;
00185
00186         /* Check spatial range... */
00187         if (atm1->p[ip] > p0 || atm1->p[ip] < p1

```

```

00188         || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00189         || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00190     continue;
00191     if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00192         || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00193         || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00194     continue;
00195
00196     /* Convert coordinates... */
00197     geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00198     geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00199     z1 = Z(atm1->p[ip]);
00200     z2 = Z(atm2->p[ip]);
00201
00202     /* Calculate absolute transport deviations... */
00203     ahtd[np] = DIST(x1, x2);
00204     avtd[np] = z1 - z2;
00205     for (iq = 0; iq < ctl.nq; iq++)
00206         aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00207
00208     /* Calculate relative transport deviations... */
00209     if (f > 4) {
00210
00211         /* Get trajectory lengths... */
00212         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00213         lh1[ip] += DIST(x0, x1);
00214         lv1[ip] += fabs(z1_old[ip] - z1);
00215
00216         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00217         lh2[ip] += DIST(x0, x2);
00218         lv2[ip] += fabs(z2_old[ip] - z2);
00219
00220         /* Get relative transport deviations... */
00221         if (lh1[ip] + lh2[ip] > 0)
00222             rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00223         if (lv1[ip] + lv2[ip] > 0)
00224             rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00225     }
00226
00227     /* Get relative transport deviations... */
00228     for (iq = 0; iq < ctl.nq; iq++)
00229         rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00230             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00231
00232     /* Save positions of air parcels... */
00233     lon1_old[ip] = atm1->lon[ip];
00234     lat1_old[ip] = atm1->lat[ip];
00235     z1_old[ip] = z1;
00236
00237     lon2_old[ip] = atm2->lon[ip];
00238     lat2_old[ip] = atm2->lat[ip];
00239     z2_old[ip] = z2;
00240
00241     /* Increment air parcel counter... */
00242     np++;
00243 }
00244
00245 /* Filter data... */
00246 if (zscore > 0 && np > 1) {
00247
00248     /* Get means and standard deviations of transport deviations... */
00249     size_t n = (size_t) np;
00250     double muh = gsl_stats_mean(ahtd, 1, n);
00251     double muv = gsl_stats_mean(avtd, 1, n);
00252     double sigh = gsl_stats_sd(ahtd, 1, n);
00253     double sigv = gsl_stats_sd(avtd, 1, n);
00254
00255     /* Filter data... */
00256     np = 0;
00257     for (size_t i = 0; i < n; i++)
00258         if (fabs(ahtd[i] - muh) / sigh < zscore
00259             && fabs(avtd[i] - muv) / sigv < zscore) {
00260             ahtd[np] = ahtd[i];
00261             rhtd[np] = rhtd[i];
00262             avtd[np] = avtd[i];
00263             rvtd[np] = rvtd[i];
00264             for (iq = 0; iq < ctl.nq; iq++) {
00265                 aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
00266                 rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
00267             }
00268             np++;
00269         }
00270 }
00271
00272 /* Get statistics... */
00273 if (strcmp(argv[3], "mean") == 0) {
00274     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);

```

```

00275     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00276     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00277     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00278     for (iq = 0; iq < ctl.nq; iq++) {
00279         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00280         rqtmd[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00281     }
00282 } else if (strcmp(argv[3], "stddev") == 0) {
00283     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00284     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00285     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00286     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00287     for (iq = 0; iq < ctl.nq; iq++) {
00288         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00289         rqtmd[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00290     }
00291 } else if (strcmp(argv[3], "min") == 0) {
00292     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00293     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00294     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00295     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00296     for (iq = 0; iq < ctl.nq; iq++) {
00297         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00298         rqtmd[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00299     }
00300 } else if (strcmp(argv[3], "max") == 0) {
00301     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00302     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00303     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00304     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00305     for (iq = 0; iq < ctl.nq; iq++) {
00306         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00307         rqtmd[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00308     }
00309 } else if (strcmp(argv[3], "skew") == 0) {
00310     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00311     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00312     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00313     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00314     for (iq = 0; iq < ctl.nq; iq++) {
00315         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00316         rqtmd[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00317     }
00318 } else if (strcmp(argv[3], "kurt") == 0) {
00319     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00320     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00321     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00322     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00323     for (iq = 0; iq < ctl.nq; iq++) {
00324         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00325         rqtmd[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00326     }
00327 } else if (strcmp(argv[3], "absdev") == 0) {
00328     ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
00329     rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
00330     avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
00331     rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
00332     for (iq = 0; iq < ctl.nq; iq++) {
00333         aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
00334         rqtmd[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
00335     }
00336 } else if (strcmp(argv[3], "median") == 0) {
00337     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00338     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00339     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00340     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00341     for (iq = 0; iq < ctl.nq; iq++) {
00342         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00343         rqtmd[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00344     }
00345 } else if (strcmp(argv[3], "mad") == 0) {
00346     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00347     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00348     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00349     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00350     for (iq = 0; iq < ctl.nq; iq++) {
00351         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00352         rqtmd[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00353     }
00354 } else
00355     ERRMSG("Unknown parameter!");
00356
00357 /* Write output... */
00358 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00359         ahtdm, rhtdm, avtdm, rvtdm);
00360 for (iq = 0; iq < ctl.nq; iq++) {
00361     fprintf(out, " ");

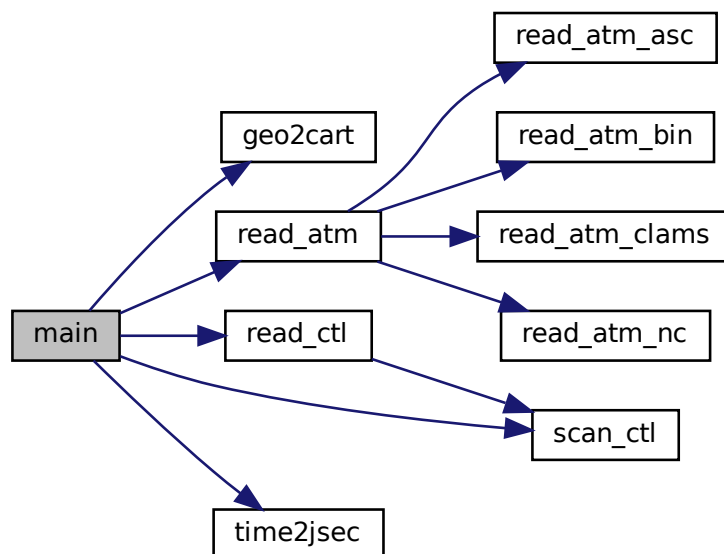
```

```

00362     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00363     fprintf(out, " ");
00364     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00365 }
00366     fprintf(out, " %d\n", np);
00367 }
00368
00369 /* Close file... */
00370 fclose(out);
00371
00372 /* Free... */
00373 free(atm1);
00374 free(atm2);
00375 free(lon1_old);
00376 free(lat1_old);
00377 free(z1_old);
00378 free(lh1);
00379 free(lv1);
00380 free(lon2_old);
00381 free(lat2_old);
00382 free(z2_old);
00383 free(lh2);
00384 free(lv2);
00385 free(ahtd);
00386 free(avtd);
00387 free(aqtd);
00388 free(rhtd);
00389 free(rvtd);
00390 free(rqtd);
00391 free(work);
00392
00393 return EXIT_SUCCESS;
00394 }

```

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) 2013-2022 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, pl, *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     pl = 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 (%) [km]\n"
00112      "# $4 = relative horizontal distance (%) [%]\n"
00113      "# $5 = absolute vertical distance (%) [km]\n"
00114      "# $6 = relative vertical distance (%) [%]\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]\n"
00119          "# %d = %s relative difference (%) [%%]\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      size_t len = strlen(argv[f]);
00137      sprintf(tstr, "%.4s", &argv[f][len - 20]);
00138      year = atoi(tstr);
00139      sprintf(tstr, "%.2s", &argv[f][len - 15]);
00140      mon = atoi(tstr);
00141      sprintf(tstr, "%.2s", &argv[f][len - 12]);
00142      day = atoi(tstr);
00143      sprintf(tstr, "%.2s", &argv[f][len - 9]);
00144      hour = atoi(tstr);
00145      sprintf(tstr, "%.2s", &argv[f][len - 6]);
00146      min = atoi(tstr);
00147      time2jsec(year, mon, day, hour, min, 0, 0, &t);
00148
00149      /* Check time... */
00150      if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00151          || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00152          ERRMSG("Cannot read time from filename!");
00153
00154      /* Save initial time... */
00155      if (!init) {
00156          init = 1;
00157          t0 = t;
00158      }
00159
00160      /* Init... */
00161      np = 0;
00162      for (ip = 0; ip < atm1->np; ip++) {
00163          ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00164          for (iq = 0; iq < ctl.nq; iq++)
00165              aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00166      }
00167
00168      /* Loop over air parcels... */
00169      for (ip = 0; ip < atm1->np; ip++) {
00170
00171          /* Check air parcel index... */
00172          if (ctl.qnt_idx > 0
00173              && (atm1->q[ctl.qnt_idx][ip] != atm2->q[ctl.qnt_idx][ip]))
00174              ERRMSG("Air parcel index does not match!");
00175
00176          /* Check ensemble index... */
00177          if (ctl.qnt_ens > 0
00178              && (atm1->q[ctl.qnt_ens][ip] != ens
00179                  || atm2->q[ctl.qnt_ens][ip] != ens))
00180              continue;
00181
00182          /* Check time... */
00183          if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00184              continue;
00185
00186          /* Check spatial range... */

```



```

00187     if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00188         || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00189         || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00190         continue;
00191     if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00192         || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00193         || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00194         continue;
00195
00196     /* Convert coordinates... */
00197     geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00198     geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00199     z1 = Z(atm1->p[ip]);
00200     z2 = Z(atm2->p[ip]);
00201
00202     /* Calculate absolute transport deviations... */
00203     ahtd[np] = DIST(x1, x2);
00204     avtd[np] = z1 - z2;
00205     for (iq = 0; iq < ctl.nq; iq++)
00206         aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00207
00208     /* Calculate relative transport deviations... */
00209     if (f > 4) {
00210
00211         /* Get trajectory lengths... */
00212         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00213         lh1[ip] += DIST(x0, x1);
00214         lv1[ip] += fabs(z1_old[ip] - z1);
00215
00216         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00217         lh2[ip] += DIST(x0, x2);
00218         lv2[ip] += fabs(z2_old[ip] - z2);
00219
00220         /* Get relative transport deviations... */
00221         if (lh1[ip] + lh2[ip] > 0)
00222             rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00223         if (lv1[ip] + lv2[ip] > 0)
00224             rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00225     }
00226
00227     /* Get relative transport deviations... */
00228     for (iq = 0; iq < ctl.nq; iq++)
00229         rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00230             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00231
00232     /* Save positions of air parcels... */
00233     lon1_old[ip] = atm1->lon[ip];
00234     lat1_old[ip] = atm1->lat[ip];
00235     z1_old[ip] = z1;
00236
00237     lon2_old[ip] = atm2->lon[ip];
00238     lat2_old[ip] = atm2->lat[ip];
00239     z2_old[ip] = z2;
00240
00241     /* Increment air parcel counter... */
00242     np++;
00243 }
00244
00245 /* Filter data... */
00246 if (zscore > 0 && np > 1) {
00247
00248     /* Get means and standard deviations of transport deviations... */
00249     size_t n = (size_t) np;
00250     double muh = gsl_stats_mean(ahtd, 1, n);
00251     double muv = gsl_stats_mean(avtd, 1, n);
00252     double sigh = gsl_stats_sd(ahtd, 1, n);
00253     double sigv = gsl_stats_sd(avtd, 1, n);
00254
00255     /* Filter data... */
00256     np = 0;
00257     for (size_t i = 0; i < n; i++)
00258         if (fabs(ahtd[i] - muh) / sigh < zscore
00259             && fabs(avtd[i] - muv) / sigv < zscore) {
00260             ahtd[np] = ahtd[i];
00261             rhtd[np] = rhtd[i];
00262             avtd[np] = avtd[i];
00263             rvtd[np] = rvtd[i];
00264             for (iq = 0; iq < ctl.nq; iq++) {
00265                 aqtd[iq * NP + np] = aqtd[iq * NP + (int) i];
00266                 rqtd[iq * NP + np] = rqtd[iq * NP + (int) i];
00267             }
00268             np++;
00269         }
00270 }
00271
00272 /* Get statistics... */
00273 if (strcasecmp(argv[3], "mean") == 0) {

```

```

00274     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00275     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00276     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00277     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00278     for (iq = 0; iq < ctl.nq; iq++) {
00279         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00280         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00281     }
00282 } else if (strcasecmp(argv[3], "stddev") == 0) {
00283     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00284     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00285     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00286     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00287     for (iq = 0; iq < ctl.nq; iq++) {
00288         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00289         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00290     }
00291 } else if (strcasecmp(argv[3], "min") == 0) {
00292     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00293     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00294     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00295     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00296     for (iq = 0; iq < ctl.nq; iq++) {
00297         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00298         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00299     }
00300 } else if (strcasecmp(argv[3], "max") == 0) {
00301     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00302     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00303     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00304     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00305     for (iq = 0; iq < ctl.nq; iq++) {
00306         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00307         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00308     }
00309 } else if (strcasecmp(argv[3], "skew") == 0) {
00310     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00311     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00312     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00313     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00314     for (iq = 0; iq < ctl.nq; iq++) {
00315         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00316         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00317     }
00318 } else if (strcasecmp(argv[3], "kurt") == 0) {
00319     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00320     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00321     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00322     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00323     for (iq = 0; iq < ctl.nq; iq++) {
00324         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00325         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00326     }
00327 } else if (strcasecmp(argv[3], "absdev") == 0) {
00328     ahtdm = gsl_stats_absdev_m(ahtd, 1, (size_t) np, 0.0);
00329     rhtdm = gsl_stats_absdev_m(rhtd, 1, (size_t) np, 0.0);
00330     avtdm = gsl_stats_absdev_m(avtd, 1, (size_t) np, 0.0);
00331     rvtdm = gsl_stats_absdev_m(rvtd, 1, (size_t) np, 0.0);
00332     for (iq = 0; iq < ctl.nq; iq++) {
00333         aqtdm[iq] = gsl_stats_absdev_m(&aqtd[iq * NP], 1, (size_t) np, 0.0);
00334         rqtdm[iq] = gsl_stats_absdev_m(&rqtd[iq * NP], 1, (size_t) np, 0.0);
00335     }
00336 } else if (strcasecmp(argv[3], "median") == 0) {
00337     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00338     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00339     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00340     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00341     for (iq = 0; iq < ctl.nq; iq++) {
00342         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00343         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00344     }
00345 } else if (strcasecmp(argv[3], "mad") == 0) {
00346     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00347     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00348     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00349     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00350     for (iq = 0; iq < ctl.nq; iq++) {
00351         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00352         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00353     }
00354 } else
00355     ERRMSG("Unknown parameter!");
00356
00357 /* Write output... */
00358 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00359         ahtdm, rhtdm, avtdm, rvtdm);
00360 for (iq = 0; iq < ctl.nq; iq++) {

```

```

00361     fprintf(out, " ");
00362     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00363     fprintf(out, " ");
00364     fprintf(out, ctl.qnt_format[iq], rqtmd[iq]);
00365 }
00366     fprintf(out, " %d\n", np);
00367 }
00368
00369 /* Close file... */
00370 fclose(out);
00371
00372 /* Free... */
00373 free(atm1);
00374 free(atm2);
00375 free(lon1_old);
00376 free(lat1_old);
00377 free(z1_old);
00378 free(lh1);
00379 free(lv1);
00380 free(lon2_old);
00381 free(lat2_old);
00382 free(z2_old);
00383 free(lh2);
00384 free(lv2);
00385 free(ahtd);
00386 free(avtd);
00387 free(aqtd);
00388 free(rhtd);
00389 free(rvtd);
00390 free(rqtd);
00391 free(work);
00392
00393 return EXIT_SUCCESS;
00394 }

```

5.5 atm_init.c File Reference

Create atmospheric data file with initial air parcel positions.

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

Functions

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

5.5.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

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

5.5.2 Function Documentation

```

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

```

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

```

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     /* Initialize air parcel index... */
00144     if (ctl.qnt_idx >= 0)
00145         for (ip = 0; ip < atm->np; ip++)
00146             atm->q[ctl.qnt_idx][ip] = ip;
00147
00148     /* Save data... */
00149     write_atm(argv[2], &ctl, atm, 0);
00150
00151     /* Free... */
00152     gsl_rng_free(rng);
00153     free(atm);
00154
00155     return EXIT_SUCCESS;
00156 }

```

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) 2013-2022 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_PLAT", -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         /* Initialize air parcel index... */
00144         if (ctl.qnt_idx >= 0)
00145             for (ip = 0; ip < atm->np; ip++)
00146                 atm->q[ctl.qnt_idx][ip] = ip;
00147
00148         /* Save data... */
00149         write_atm(argv[2], &ctl, atm, 0);
00150
00151         /* Free... */
00152         gsl_rng_free(rng);
00153         free(atm);
00154
00155         return EXIT_SUCCESS;
00156     }

```

5.7 atm_select.c File Reference

Extract subsets of air parcels from atmospheric data files.

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

Functions

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

5.7.1 Detailed Description

Extract subsets of air parcels from atmospheric data files.

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

5.7.2 Function Documentation

```

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

```

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

```

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, idx0, idx1, 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         idx0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX0", -1, "-999", NULL);
00053         idx1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX1", -1, "-999", NULL);
00054         ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00055         ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00056         t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00057         t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00058         p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00059         p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00060         lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00061         lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00062         lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00063         lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00064         r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00065         r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00066         rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00067         rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00068
00069         /* Get Cartesian coordinates... */
00070         geo2cart(0, rlon, rlat, x0);
00071
00072         /* Loop over files... */
00073         for (f = 3; f < argc; f++) {
00074
00075             /* Read atmospheric data... */
00076             if (!read_atm(argv[f], &ctl, atm))
00077                 continue;
00078
00079             /* Adjust range of air parcels... */
00080             if (ip0 < 0)
00081                 ip0 = 0;
00082             ip0 = GSL_MIN(ip0, atm->np - 1);
00083             if (ip1 < 0)
00084                 ip1 = atm->np - 1;
00085             ip1 = GSL_MIN(ip1, atm->np - 1);
00086             if (ip1 < ip0)
00087                 ip1 = ip0;
00088
00089             /* Loop over air parcels... */
00090             for (ip = ip0; ip <= ip1; ip += stride) {
00091
00092                 /* Check air parcel index... */
00093                 if (ctl.qnt_idx >= 0 && idx0 >= 0 && idx1 >= 0)
00094                     if (atm->q[ctl.qnt_idx][ip] < idx0 || atm->q[ctl.qnt_idx][ip] > idx1)
00095                         continue;
00096
00097                 /* Check time... */
00098                 if (t0 != t1)
00099                     if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00100                         || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00101                         continue;
00102
00103                 /* Check vertical distance... */
00104                 if (p0 != p1)
00105                     if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00106                         || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00107                         continue;
00108

```

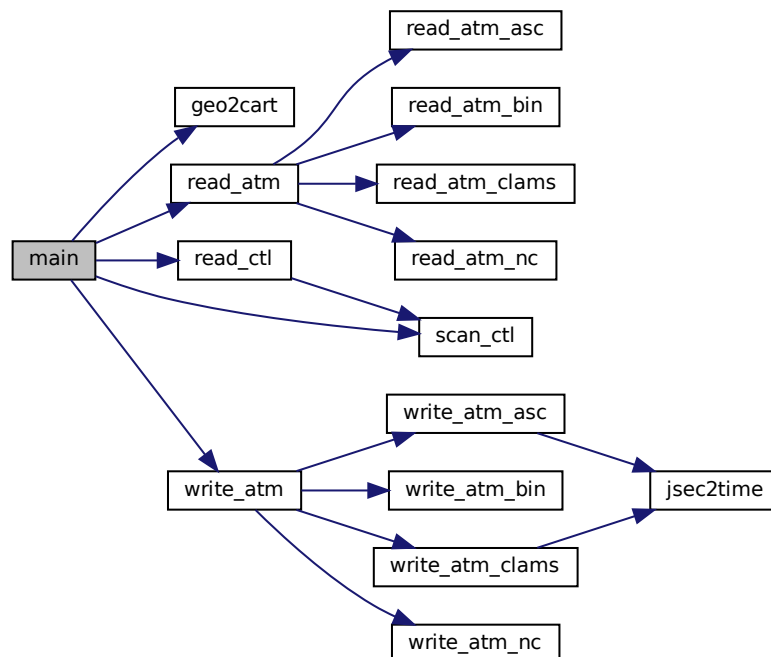


```

00109     /* Check longitude... */
00110     if (lon0 != lon1)
00111         if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00112             || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00113         continue;
00114
00115     /* Check latitude... */
00116     if (lat0 != lat1)
00117         if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00118             || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00119         continue;
00120
00121     /* Check horizontal distace... */
00122     if (r0 != r1) {
00123         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00124         r = DIST(x0, x1);
00125         if ((r1 > r0 && (r < r0 || r > r1))
00126             || (r1 < r0 && (r < r0 && r > r1)))
00127             continue;
00128     }
00129
00130     /* Copy data... */
00131     atm2->time[atm2->np] = atm->time[ip];
00132     atm2->p[atm2->np] = atm->p[ip];
00133     atm2->lon[atm2->np] = atm->lon[ip];
00134     atm2->lat[atm2->np] = atm->lat[ip];
00135     for (iq = 0; iq < ctl.nq; iq++)
00136         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00137     if ((++atm2->np) > NP)
00138         ERRMSG("Too many air parcels!");
00139 }
00140 }
00141
00142 /* Close file... */
00143 write_atm(argv[2], &ctl, atm2, 0);
00144
00145 /* Free... */
00146 free(atm);
00147 free(atm2);
00148
00149 return EXIT_SUCCESS;
00150 }

```

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) 2013-2022 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, idx0, idx1, 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     idx0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX0", -1, "-999", NULL);
00053     idx1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IDX1", -1, "-999", NULL);
00054     ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "-999", NULL);
00055     ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "-999", NULL);
00056     t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00057     t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00058     p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00059     p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00060     lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00061     lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00062     lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00063     lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00064     r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00065     r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00066     rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00067     rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00068
00069     /* Get Cartesian coordinates... */
00070     geo2cart(0, rlon, rlat, x0);
00071
00072     /* Loop over files... */
00073     for (f = 3; f < argc; f++) {
00074
00075         /* Read atmopheric data... */
00076         if (!read_atm(argv[f], &ctl, atm))
00077             continue;
00078
00079         /* Adjust range of air parcels... */
00080         if (ip0 < 0)
00081             ip0 = 0;
00082         ip0 = GSL_MIN(ip0, atm->np - 1);
00083         if (ip1 < 0)
00084             ip1 = atm->np - 1;
00085         ip1 = GSL_MIN(ip1, atm->np - 1);
00086         if (ip1 < ip0)
00087             ip1 = ip0;
00088
00089         /* Loop over air parcels... */
00090         for (ip = ip0; ip <= ip1; ip += stride) {

```

```

00091
00092     /* Check air parcel index... */
00093     if (ctl.qnt_idx >= 0 && idx0 >= 0 && idx1 >= 0)
00094         if (atm->q[ctl.qnt_idx][ip] < idx0 || atm->q[ctl.qnt_idx][ip] > idx1)
00095             continue;
00096
00097     /* Check time... */
00098     if (t0 != t1)
00099         if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00100             || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00101             continue;
00102
00103     /* Check vertical distance... */
00104     if (p0 != p1)
00105         if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00106             || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00107             continue;
00108
00109     /* Check longitude... */
00110     if (lon0 != lon1)
00111         if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00112             || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00113             continue;
00114
00115     /* Check latitude... */
00116     if (lat0 != lat1)
00117         if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00118             || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00119             continue;
00120
00121     /* Check horizontal distace... */
00122     if (r0 != r1) {
00123         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00124         r = DIST(x0, x1);
00125         if ((r1 > r0 && (r < r0 || r > r1))
00126             || (r1 < r0 && (r < r0 && r > r1)))
00127             continue;
00128     }
00129
00130     /* Copy data... */
00131     atm2->time[atm2->np] = atm->time[ip];
00132     atm2->p[atm2->np] = atm->p[ip];
00133     atm2->lon[atm2->np] = atm->lon[ip];
00134     atm2->lat[atm2->np] = atm->lat[ip];
00135     for (iq = 0; iq < ctl.nq; iq++)
00136         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00137     if ((++atm2->np) > NP)
00138         ERRMSG("Too many air parcels!");
00139 }
00140 }
00141
00142 /* Close file... */
00143 write_atm(argv[2], &ctl, atm2, 0);
00144
00145 /* Free... */
00146 free(atm);
00147 free(atm2);
00148
00149 return EXIT_SUCCESS;
00150 }

```

5.9 atm_split.c File Reference

Split air parcels into a larger number of parcels.

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

Functions

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

5.9.1 Detailed Description

Split air parcels into a larger number of parcels.

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

5.9.2 Function Documentation

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

Definition at line 27 of file `atm_split.c`.

```
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[EP], kz[EP], 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 >= EP)
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     /* Select air parcel... */
00119     if (ctl.qnt_m >= 0)
00120         do {
00121             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00122         } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00123     else
00124         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00125
00126     /* Set time... */
00127     if (t1 > t0)
00128         atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00129     else
00130         atm2->time[atm2->np] = atm->time[ip]
00131             + gsl_rng_gaussian_ziggurat(rng, dt / 2.3548);
00132
00133     /* Set vertical position... */
00134     do {
00135         if (nz > 0) {
00136             do {
00137                 z = zmin + (zmax - zmin) * gsl_rng_uniform_pos(rng);
00138                 iz = locate_irr(kz, nz, z);
00139                 k = LIN(kz[iz], kk[iz], kz[iz + 1], kk[iz + 1], z);
00140             } while (gsl_rng_uniform(rng) > k);
00141             atm2->p[atm2->np] = P(z);
00142         } else if (z1 > z0)
00143             atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00144         else
00145             atm2->p[atm2->np] = atm->p[ip]
00146                 + DZ2DP(gsl_rng_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00147     } while (atm2->p[atm2->np] < P(100.) || atm2->p[atm2->np] > P(-1.));
00148
00149     /* Set horizontal position... */
00150     if (lon1 > lon0 && lat1 > lat0) {
00151         atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00152         atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00153     } else {
00154         atm2->lon[atm2->np] = atm->lon[ip]
00155             + gsl_rng_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00156         atm2->lat[atm2->np] = atm->lat[ip]
00157             + gsl_rng_gaussian_ziggurat(rng, DY2DEG(dy, atm->lat[ip]) / 2.3548);
00158     }
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 /* Adjust air parcel index... */
00170 if (ctl.qnt_idx >= 0)
00171     atm2->q[ctl.qnt_idx][atm2->np] = atm2->np;
00172
00173 /* Increment particle counter... */
00174 if ((++atm2->np) > NP)
00175     ERRMSG("Too many air parcels!");
00176 }
00177
00178 /* Save data and close file... */
00179 write_atm(argv[3], &ctl, atm2, 0);
00180
00181 /* Free... */
00182 free(atm);
00183 free(atm2);
00184
00185 return EXIT_SUCCESS;
00186 }

```

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) 2013-2022 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     atm_t *atm, *atm2;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     FILE *in;
00038
00039     char kernel[LEN], line[LEN];
00040
00041     double dt, dx, dz, k, kk[EP], kz[EP], 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) >= EP)
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         /* Adjust air parcel index... */
00170         if (ctl.qnt_idx >= 0)
00171             atm2->q[ctl.qnt_idx][atm2->np] = atm2->np;
00172
00173         /* Increment particle counter... */
00174         if ((++atm2->np) > NP)
00175             ERRMSG("Too many air parcels!");
00176     }
00177

```

```
00178  /* Save data and close file... */
00179  write_atm(argv[3], &ctl, atm2, 0);
00180
00181  /* Free... */
00182  free(atm);
00183  free(atm2);
00184
00185  return EXIT_SUCCESS;
00186 }
```

5.11 atm_stat.c File Reference

Calculate air parcel statistics.

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

Functions

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

5.11.1 Detailed Description

Calculate air parcel statistics.

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

5.11.2 Function Documentation

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

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

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



```

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 LOG(1, "Write air parcel statistics: %s", 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, "# $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     size_t len = strlen(argv[f]);
00094     sprintf(tstr, "%.4s", &argv[f][len - 20]);
00095     year = atoi(tstr);
00096     sprintf(tstr, "%.2s", &argv[f][len - 15]);
00097     mon = atoi(tstr);
00098     sprintf(tstr, "%.2s", &argv[f][len - 12]);
00099     day = atoi(tstr);
00100     sprintf(tstr, "%.2s", &argv[f][len - 9]);
00101     hour = atoi(tstr);
00102     sprintf(tstr, "%.2s", &argv[f][len - 6]);
00103     min = atoi(tstr);
00104     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00105
00106     /* Check time... */
00107     if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00108         || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00109         ERRMSG("Cannot read time from filename!");
00110
00111     /* Save initial time... */
00112     if (!init) {
00113         init = 1;
00114         t0 = t;
00115     }
00116
00117     /* Filter data... */
00118     atm_filt->np = 0;
00119     for (ip = 0; ip < atm->np; ip++) {
00120
00121         /* Check time... */
00122         if (!gsl_finite(atm->time[ip]))
00123             continue;
00124
00125         /* Check ensemble index... */
00126         if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00127             continue;
00128
00129         /* Check spatial range... */
00130         if (atm->p[ip] > p0 || atm->p[ip] < p1
00131             || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00132             || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00133             continue;
00134
00135         /* Save data... */
00136         atm_filt->time[atm_filt->np] = atm->time[ip];
00137         atm_filt->p[atm_filt->np] = atm->p[ip];
00138         atm_filt->lon[atm_filt->np] = atm->lon[ip];
00139         atm_filt->lat[atm_filt->np] = atm->lat[ip];
00140         for (iq = 0; iq < ctl.nq; iq++)
00141             atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];

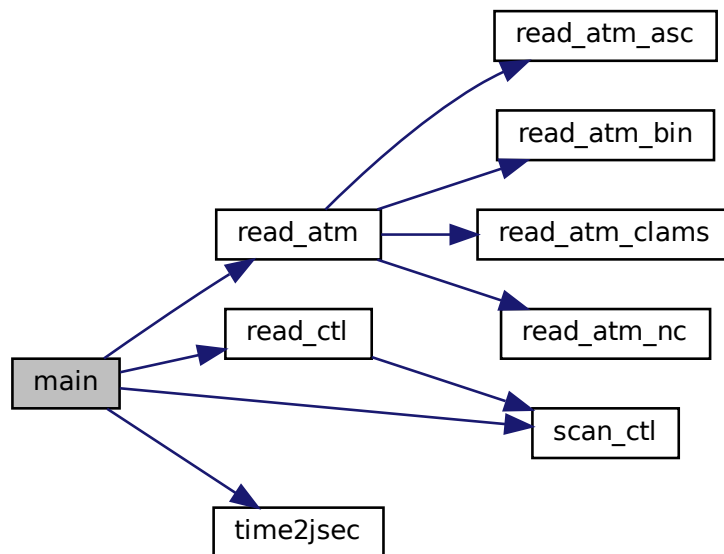
```

```

00142     atm_filt->np++;
00143 }
00144
00145 /* Get heights... */
00146 for (ip = 0; ip < atm_filt->np; ip++)
00147     zs[ip] = Z(atm_filt->p[ip]);
00148
00149 /* Get statistics... */
00150 if (strcasecmp(argv[3], "mean") == 0) {
00151     zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00152     lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00153     latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00154     for (iq = 0; iq < ctl.nq; iq++)
00155         qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156 } else if (strcasecmp(argv[3], "stddev") == 0) {
00157     zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00158     lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00159     latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00160     for (iq = 0; iq < ctl.nq; iq++)
00161         qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162 } else if (strcasecmp(argv[3], "min") == 0) {
00163     zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00164     lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00165     latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00166     for (iq = 0; iq < ctl.nq; iq++)
00167         qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168 } else if (strcasecmp(argv[3], "max") == 0) {
00169     zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00170     lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00171     latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00172     for (iq = 0; iq < ctl.nq; iq++)
00173         qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174 } else if (strcasecmp(argv[3], "skew") == 0) {
00175     zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00176     lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00177     latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00178     for (iq = 0; iq < ctl.nq; iq++)
00179         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00180 } else if (strcasecmp(argv[3], "kurt") == 0) {
00181     zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00182     lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00183     latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00184     for (iq = 0; iq < ctl.nq; iq++)
00185         qm[iq] =
00186             gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187 } else if (strcasecmp(argv[3], "median") == 0) {
00188     zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00189     lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00190     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00191     for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193 } else if (strcasecmp(argv[3], "absdev") == 0) {
00194     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00195     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00196     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00197     for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00199 } else if (strcasecmp(argv[3], "mad") == 0) {
00200     zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00201     lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00202     latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00203     for (iq = 0; iq < ctl.nq; iq++)
00204         qm[iq] =
00205             gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00206 } else
00207     ERRMSG("Unknown parameter!");
00208
00209 /* Write data... */
00210 fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00211 for (iq = 0; iq < ctl.nq; iq++) {
00212     fprintf(out, " ");
00213     fprintf(out, ctl.qnt_format[iq], qm[iq]);
00214 }
00215 fprintf(out, "\n", atm_filt->np);
00216 }
00217
00218 /* Close file... */
00219 fclose(out);
00220
00221 /* Free... */
00222 free(atm);
00223 free(atm_filt);
00224 free(work);
00225 free(zs);
00226
00227 return EXIT_SUCCESS;
00228 }

```

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) 2013-2022 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     NP);
00049     ALLOC(zs, double,
00050     NP);
00051
00052     /* Check arguments... */
00053     if (argc < 4)
00054         ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atml> [<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     LOG(1, "Write air parcel statistics: %s", 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, "# $%d = %s (%) [%s]\n", iq + 6,
00082                 ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083     fprintf(out, "# $%d = number of particles\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         size_t len = strlen(argv[f]);
00094         sprintf(tstr, "%.4s", &argv[f][len - 20]);
00095         year = atoi(tstr);
00096         sprintf(tstr, "%.2s", &argv[f][len - 15]);
00097         mon = atoi(tstr);
00098         sprintf(tstr, "%.2s", &argv[f][len - 12]);
00099         day = atoi(tstr);
00100         sprintf(tstr, "%.2s", &argv[f][len - 9]);
00101         hour = atoi(tstr);
00102         sprintf(tstr, "%.2s", &argv[f][len - 6]);
00103         min = atoi(tstr);
00104         time2jsec(year, mon, day, hour, min, 0, 0, &t);
00105
00106         /* Check time... */
00107         if (year < 1900 || year > 2100 || mon < 1 || mon > 12 || day < 1
00108             || day > 31 || hour < 0 || hour > 23 || min < 0 || min > 59)
00109             ERRMSG("Cannot read time from filename!");
00110
00111         /* Save initial time... */
00112         if (!init) {
00113             init = 1;
00114             t0 = t;
00115         }
00116
00117         /* Filter data... */
00118         atm_filt->np = 0;
00119         for (ip = 0; ip < atm->np; ip++) {
00120
00121             /* Check time... */
00122             if (!gsl_finite(atm->time[ip]))
00123                 continue;
00124
00125             /* Check ensemble index... */
00126             if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00127                 continue;
00128
00129             /* Check spatial range... */
00130             if (atm->p[ip] > p0 || atm->p[ip] < p1
00131                 || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00132                 || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00133                 continue;
00134

```

```

00135     /* Save data... */
00136     atm_filt->time[atm_filt->np] = atm->time[ip];
00137     atm_filt->p[atm_filt->np] = atm->p[ip];
00138     atm_filt->lon[atm_filt->np] = atm->lon[ip];
00139     atm_filt->lat[atm_filt->np] = atm->lat[ip];
00140     for (iq = 0; iq < ctl.nq; iq++)
00141         atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00142     atm_filt->np++;
00143 }
00144
00145 /* Get heights... */
00146 for (ip = 0; ip < atm_filt->np; ip++)
00147     zs[ip] = Z(atm_filt->p[ip]);
00148
00149 /* Get statistics... */
00150 if (strcasecmp(argv[3], "mean") == 0) {
00151     zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00152     lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00153     latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00154     for (iq = 0; iq < ctl.nq; iq++)
00155         qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156 } else if (strcasecmp(argv[3], "stddev") == 0) {
00157     zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00158     lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00159     latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00160     for (iq = 0; iq < ctl.nq; iq++)
00161         qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162 } else if (strcasecmp(argv[3], "min") == 0) {
00163     zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00164     lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00165     latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00166     for (iq = 0; iq < ctl.nq; iq++)
00167         qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168 } else if (strcasecmp(argv[3], "max") == 0) {
00169     zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00170     lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00171     latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00172     for (iq = 0; iq < ctl.nq; iq++)
00173         qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174 } else if (strcasecmp(argv[3], "skew") == 0) {
00175     zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00176     lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00177     latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00178     for (iq = 0; iq < ctl.nq; iq++)
00179         qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00180 } else if (strcasecmp(argv[3], "kurt") == 0) {
00181     zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00182     lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00183     latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00184     for (iq = 0; iq < ctl.nq; iq++)
00185         qm[iq] =
00186             gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187 } else if (strcasecmp(argv[3], "median") == 0) {
00188     zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00189     lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00190     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00191     for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193 } else if (strcasecmp(argv[3], "absdev") == 0) {
00194     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00195     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00196     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00197     for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00199 } else if (strcasecmp(argv[3], "mad") == 0) {
00200     zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00201     lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00202     latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00203     for (iq = 0; iq < ctl.nq; iq++)
00204         qm[iq] =
00205             gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00206 } else
00207     ERRMSG("Unknown parameter!");
00208
00209 /* Write data... */
00210 fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00211 for (iq = 0; iq < ctl.nq; iq++) {
00212     fprintf(out, " ");
00213     fprintf(out, ctl.qnt_format[iq], qm[iq]);
00214 }
00215 fprintf(out, " %d\n", atm_filt->np);
00216 }
00217
00218 /* Close file... */
00219 fclose(out);
00220
00221 /* Free... */

```

```
00222     free(atm);
00223     free(atm_filt);
00224     free(work);
00225     free(zs);
00226
00227     return EXIT_SUCCESS;
00228 }
```

5.13 cape.c File Reference

Add CAPE data to netCDF file.

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

Functions

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

5.13.1 Detailed Description

Add CAPE data to netCDF file.

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

5.13.2 Function Documentation

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

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

```
00029     {
00030
00031     ctl_t ctl;
00032
00033     clim_t *clim;
00034
00035     met_t *met;
00036
00037     char tstr[LEN];
00038
00039     float help[EX * EY];
00040
00041     int dims[10], ncid, varid;
00042
00043     size_t start[10], count[10];
00044
00045     /* Allocate... */
00046     ALLOC(clim, clim_t, 1);
00047     ALLOC(met, met_t, 1);
00048
00049     /* Check arguments... */
00050     if (argc < 2)
00051         ERRMSG("Give parameters: <ctl> <met.nc>");
00052
00053     /* Read control parameters... */
00054     read_ctl(argv[1], argc, argv, &ctl);
00055
00056     /* Read climatological data... */
```

```

00057     read_clim(&ctl, clim);
00058
00059     /* Read meteorological data... */
00060     if (!read_met(argv[2], &ctl, clim, met))
00061         ERRMSG("Cannot open file!");
00062
00063     /* Open netCDF file... */
00064     if (nc_open(argv[2], NC_WRITE, &ncid) != NC_NOERR)
00065         ERRMSG("Cannot open file!");
00066
00067     /* Get dimensions... */
00068     NC_INQ_DIM("time", &dims[0], 1, 1);
00069     NC_INQ_DIM("lat", &dims[1], met->ny, met->ny);
00070     NC_INQ_DIM("lon", &dims[2], met->nx - 1, met->nx - 1);
00071     NC(nc_inq_dimid(ncid, "time", &dims[0]));
00072     NC(nc_inq_dimid(ncid, "lat", &dims[1]));
00073     NC(nc_inq_dimid(ncid, "lon", &dims[2]));
00074
00075     /* Set define mode... */
00076     NC(nc_redef(ncid));
00077
00078     /* Create variables... */
00079     NC_DEF_VAR("CAPE_MPT", NC_FLOAT, 3, dims,
00080               "convective available potential energy", "J kg**-1");
00081     NC_DEF_VAR("CIN_MPT", NC_FLOAT, 3, dims,
00082               "convective inhibition", "J kg**-1");
00083     NC_DEF_VAR("PEL_MPT", NC_FLOAT, 3, dims,
00084               "pressure at equilibrium level", "hPa");
00085
00086     /* Get current time... */
00087     time_t t = time(NULL);
00088     struct tm tm = *localtime(&t);
00089     sprintf(tstr, "%d-%02d-%02d %02d:%02d",
00090             tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
00091             tm.tm_hour, tm.tm_min, tm.tm_sec);
00092
00093     /* Set additional attributes... */
00094     NC_PUT_ATT("CAPE_MPT", "creator_of_parameter", "MPTRAC");
00095     NC_PUT_ATT("CIN_MPT", "creator_of_parameter", "MPTRAC");
00096     NC_PUT_ATT("PEL_MPT", "creator_of_parameter", "MPTRAC");
00097
00098     NC_PUT_ATT("CAPE_MPT", "param_creation_time", tstr);
00099     NC_PUT_ATT("CIN_MPT", "param_creation_time", tstr);
00100     NC_PUT_ATT("PEL_MPT", "param_creation_time", tstr);
00101
00102     NC_PUT_ATT("CAPE_MPT", "param_modification_time", tstr);
00103     NC_PUT_ATT("CIN_MPT", "param_modification_time", tstr);
00104     NC_PUT_ATT("PEL_MPT", "param_modification_time", tstr);
00105
00106     NC_PUT_ATT("CAPE_MPT", "flag", "NONE");
00107     NC_PUT_ATT("CIN_MPT", "flag", "NONE");
00108     NC_PUT_ATT("PEL_MPT", "flag", "NONE");
00109
00110     float miss[1] = { GSL_NAN };
00111     NC(nc_inq_varid(ncid, "CAPE_MPT", &varid));
00112     NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
00113     NC(nc_inq_varid(ncid, "CIN_MPT", &varid));
00114     NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
00115     NC(nc_inq_varid(ncid, "PEL_MPT", &varid));
00116     NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
00117
00118     /* End define mode... */
00119     NC(nc_enddef(ncid));
00120
00121     /* Write data... */
00122     for (int ix = 0; ix < met->nx - 1; ix++)
00123         for (int iy = 0; iy < met->ny; iy++)
00124             help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cape[ix][iy];
00125     NC_PUT_FLOAT("CAPE_MPT", help, 0);
00126
00127     for (int ix = 0; ix < met->nx - 1; ix++)
00128         for (int iy = 0; iy < met->ny; iy++)
00129             help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cin[ix][iy];
00130     NC_PUT_FLOAT("CIN_MPT", help, 0);
00131
00132     for (int ix = 0; ix < met->nx - 1; ix++)
00133         for (int iy = 0; iy < met->ny; iy++)
00134             help[ARRAY_2D(iy, ix, met->nx - 1)] = met->pel[ix][iy];
00135     NC_PUT_FLOAT("PEL_MPT", help, 0);
00136
00137     /* Close file... */
00138     nc_close(ncid);
00139
00140     /* Free... */
00141     free(clim);
00142     free(met);
00143

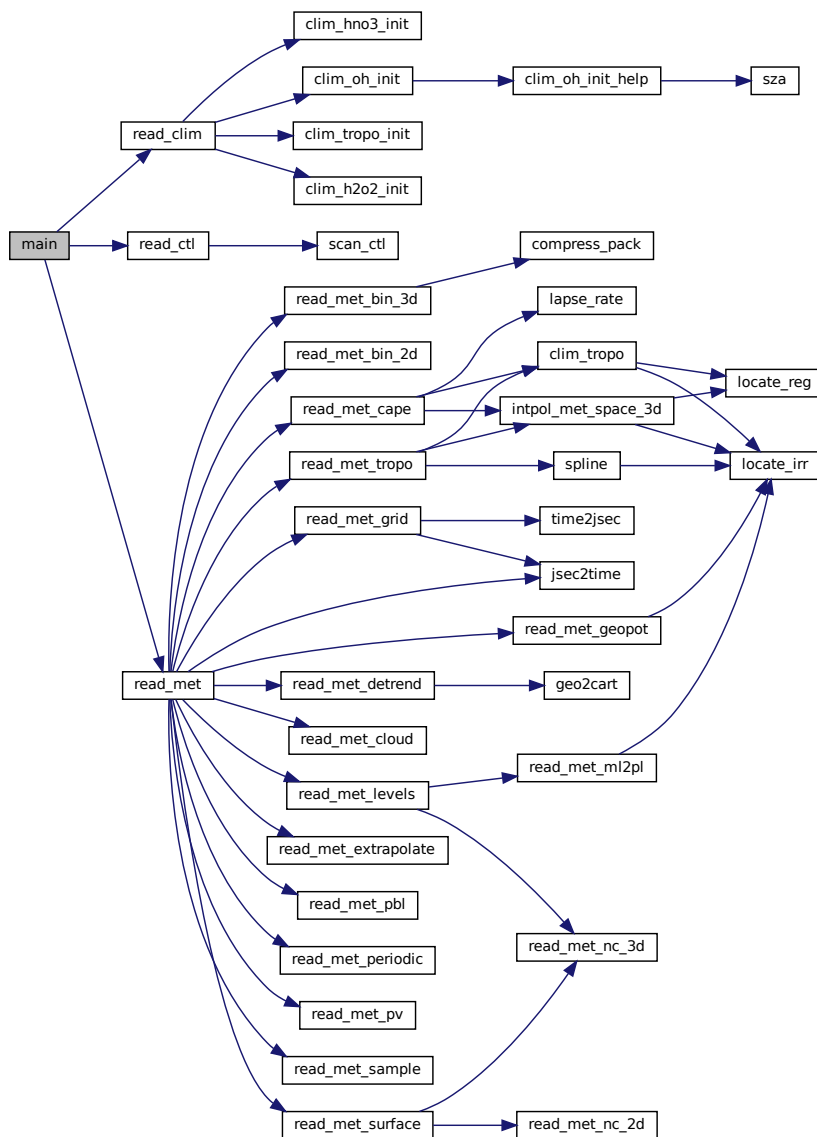
```

```

00144     return EXIT_SUCCESS;
00145 }

```

Here is the call graph for this function:



5.14 cape.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) 2013-2022 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 int main(
00023     int argc,
00024     char *argv[]) {
00025
00026     ctl_t ctl;
00027
00028     clim_t *clim;
00029
00030     met_t *met;
00031
00032     char tstr[LEN];
00033
00034     float help[EX * EY];
00035
00036     int dims[10], ncid, varid;
00037
00038     size_t start[10], count[10];
00039
00040     /* Allocate... */
00041     ALLOC(clim, clim_t, 1);
00042     ALLOC(met, met_t, 1);
00043
00044     /* Check arguments... */
00045     if (argc < 2)
00046         ERRMSG("Give parameters: <ctl> <met.nc>");
00047
00048     /* Read control parameters... */
00049     read_ctl(argv[1], argc, argv, &ctl);
00050
00051     /* Read climatological data... */
00052     read_clim(&ctl, clim);
00053
00054     /* Read meteorological data... */
00055     if (!read_met(argv[2], &ctl, clim, met))
00056         ERRMSG("Cannot open file!");
00057
00058     /* Open netCDF file... */
00059     if (nc_open(argv[2], NC_WRITE, &ncid) != NC_NOERR)
00060         ERRMSG("Cannot open file!");
00061
00062     /* Get dimensions... */
00063     NC_INQ_DIM("time", &dims[0], 1, 1);
00064     NC_INQ_DIM("lat", &dims[1], met->ny, met->ny);
00065     NC_INQ_DIM("lon", &dims[2], met->nx - 1, met->nx - 1);
00066     NC(nc_inq_dimid(ncid, "time", &dims[0]));
00067     NC(nc_inq_dimid(ncid, "lat", &dims[1]));
00068     NC(nc_inq_dimid(ncid, "lon", &dims[2]));
00069
00070     /* Set define mode... */
00071     NC(nc_redef(ncid));
00072
00073     /* Create variables... */
00074     NC_DEF_VAR("CAPE_MPT", NC_FLOAT, 3, dims,
00075         "convective available potential energy", "J kg**-1");
00076     NC_DEF_VAR("CIN_MPT", NC_FLOAT, 3, dims,
00077         "convective inhibition", "J kg**-1");
00078     NC_DEF_VAR("PEL_MPT", NC_FLOAT, 3, dims,
00079         "pressure at equilibrium level", "hPa");
00080
00081     /* Get current time... */
00082     time_t t = time(NULL);
00083     struct tm tm = *localtime(&t);
00084     sprintf(tstr, "%d-%02d-%02d %02d:%02d:%02d",
00085         tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
00086         tm.tm_hour, tm.tm_min, tm.tm_sec);
00087
00088     /* Set additional attributes... */
00089     NC_PUT_ATT("CAPE_MPT", "creator_of_parameter", "MPTRAC");
00090     NC_PUT_ATT("CIN_MPT", "creator_of_parameter", "MPTRAC");
00091     NC_PUT_ATT("PEL_MPT", "creator_of_parameter", "MPTRAC");
00092
00093     NC_PUT_ATT("CAPE_MPT", "param_creation_time", tstr);
00094     NC_PUT_ATT("CIN_MPT", "param_creation_time", tstr);
00095     NC_PUT_ATT("PEL_MPT", "param_creation_time", tstr);
00096
00097     NC_PUT_ATT("CAPE_MPT", "param_modification_time", tstr);
00098     NC_PUT_ATT("CIN_MPT", "param_modification_time", tstr);
00099     NC_PUT_ATT("PEL_MPT", "param_modification_time", tstr);
00100
00101     NC_PUT_ATT("CAPE_MPT", "flag", "NONE");
00102     NC_PUT_ATT("CIN_MPT", "flag", "NONE");

```

```

00108  NC_PUT_ATT("PEL_MPT", "flag", "NONE");
00109
00110  float miss[1] = { GSL_NAN };
00111  NC(nc_inq_varid(ncid, "CAPE_MPT", &varid));
00112  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
00113  NC(nc_inq_varid(ncid, "CIN_MPT", &varid));
00114  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
00115  NC(nc_inq_varid(ncid, "PEL_MPT", &varid));
00116  NC(nc_put_att_float(ncid, varid, "missing_value", NC_FLOAT, 1, miss));
00117
00118  /* End define mode... */
00119  NC(nc_enddef(ncid));
00120
00121  /* Write data... */
00122  for (int ix = 0; ix < met->nx - 1; ix++)
00123      for (int iy = 0; iy < met->ny; iy++)
00124          help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cape[ix][iy];
00125  NC_PUT_FLOAT("CAPE_MPT", help, 0);
00126
00127  for (int ix = 0; ix < met->nx - 1; ix++)
00128      for (int iy = 0; iy < met->ny; iy++)
00129          help[ARRAY_2D(iy, ix, met->nx - 1)] = met->cin[ix][iy];
00130  NC_PUT_FLOAT("CIN_MPT", help, 0);
00131
00132  for (int ix = 0; ix < met->nx - 1; ix++)
00133      for (int iy = 0; iy < met->ny; iy++)
00134          help[ARRAY_2D(iy, ix, met->nx - 1)] = met->pel[ix][iy];
00135  NC_PUT_FLOAT("PEL_MPT", help, 0);
00136
00137  /* Close file... */
00138  nc_close(ncid);
00139
00140  /* Free... */
00141  free(clim);
00142  free(met);
00143
00144  return EXIT_SUCCESS;
00145 }

```

5.15 day2doy.c File Reference

Convert date to day of year.

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

Functions

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

5.15.1 Detailed Description

Convert date to day of year.

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

5.15.2 Function Documentation

```

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

```

Definition at line 27 of file [day2doy.c](#).

```

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

```

Here is the call graph for this function:



5.16 day2doy.c

```

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

```

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

5.17 doy2day.c File Reference

Convert day of year to date.

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

Functions

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

5.17.1 Detailed Description

Convert day of year to date.

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

5.17.2 Function Documentation

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

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

```
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:



5.18 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 }

```

5.19 jsec2time.c File Reference

Convert Julian seconds to date.

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

Functions

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

5.19.1 Detailed Description

Convert Julian seconds to date.

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

5.19.2 Function Documentation

```

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

```

Definition at line 27 of file [jsec2time.c](#).

```

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

```

Here is the call graph for this function:



5.20 jsec2time.c

```

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

```

```

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 }

```

5.21 libtrac.c File Reference

MPTRAC library definitions.

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

Functions

- double [buoyancy_frequency](#) (double p0, double t0, double p1, double t1)
Calculate buoyancy frequency.
- void [cart2geo](#) (double *x, double *z, double *lon, double *lat)
Convert Cartesian coordinates to geolocation.
- double [clim_hno3](#) (clim_t *clim, double t, double lat, double p)
Climatology of HNO3 volume mixing ratios.
- void [clim_hno3_init](#) (clim_t *clim)
Initialization function for HNO3 climatology.
- double [clim_oh](#) (clim_t *clim, double t, double lat, double p)
Climatology of OH number concentrations.
- double [clim_oh_diurnal](#) (ctl_t *ctl, clim_t *clim, double t, double p, double lon, double lat)
Climatology of OH number concentrations with diurnal variation.
- void [clim_oh_init](#) (ctl_t *ctl, clim_t *clim)
Initialization function for OH climatology.
- double [clim_oh_init_help](#) (double beta, double time, double lat)
Apply diurnal correction to OH climatology.
- double [clim_h2o2](#) (clim_t *clim, double t, double lat, double p)
Climatology of H2O2 number concentrations.
- void [clim_h2o2_init](#) (ctl_t *ctl, clim_t *clim)
Initialization function for H2O2 climatology.
- double [clim_tropo](#) (clim_t *clim, double t, double lat)
Climatology of tropopause pressure.
- void [clim_tropo_init](#) (clim_t *clim)
Initialize tropopause climatology.
- void [compress_pack](#) (char *varname, float *array, size_t nxy, size_t nz, int decompress, FILE *inout)
Pack or unpack array.
- void [day2doy](#) (int year, int mon, int day, int *doy)
Compress or decompress array with zfp.
- void [doy2day](#) (int year, int doy, int *mon, int *day)
Get date from day of year.
- void [geo2cart](#) (double z, double lon, double lat, double *x)
Convert geolocation to Cartesian coordinates.
- void [get_met](#) (ctl_t *ctl, clim_t *clim, double t, met_t **met0, met_t **met1)
Get meteo data for given time step.
- void [get_met_help](#) (ctl_t *ctl, double t, int direct, char *metbase, double dt_met, char *filename)
Get meteo data for time step.

- void [get_met_replace](#) (char *orig, char *search, char *repl)
Replace template strings in filename.
- void [intpol_met_space_3d](#) (met_t *met, float array[EX][EY][EP], double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteo data.
- void [intpol_met_space_2d](#) (met_t *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteo data.
- void [intpol_met_time_3d](#) (met_t *met0, float array0[EX][EY][EP], met_t *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteo data.
- void [intpol_met_time_2d](#) (met_t *met0, float array0[EX][EY], met_t *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteo data.
- void [jsec2time](#) (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)
Temporal interpolation of meteo data.
- double [lapse_rate](#) (double t, double h2o)
Calculate moist adiabatic lapse rate.
- int [locate_irr](#) (double *xx, int n, double x)
Find array index for irregular grid.
- int [locate_reg](#) (double *xx, int n, double x)
Find array index for regular grid.
- double [nat_temperature](#) (double p, double h2o, double hno3)
Calculate NAT existence temperature.
- void [quicksort](#) (double arr[], int brr[], int low, int high)
Parallel quicksort.
- int [quicksort_partition](#) (double arr[], int brr[], int low, int high)
Partition function for quicksort.
- int [read_atm](#) (const char *filename, ctl_t *ctl, atm_t *atm)
Read atmospheric data.
- int [read_atm_asc](#) (const char *filename, ctl_t *ctl, atm_t *atm)
Read atmospheric data in ASCII format.
- int [read_atm_bin](#) (const char *filename, ctl_t *ctl, atm_t *atm)
Read atmospheric data in binary format.
- int [read_atm_clams](#) (const char *filename, ctl_t *ctl, atm_t *atm)
Read atmospheric data in CLaMS format.
- int [read_atm_nc](#) (const char *filename, ctl_t *ctl, atm_t *atm)
Read atmospheric data in netCDF format.
- void [read_clim](#) (ctl_t *ctl, clim_t *clim)
Read climatological data.
- void [read_ctl](#) (const char *filename, int argc, char *argv[], ctl_t *ctl)
Read control parameters.
- int [read_met](#) (char *filename, ctl_t *ctl, clim_t *clim, met_t *met)
Read meteo data file.
- void [read_met_bin_2d](#) (FILE *in, met_t *met, float var[EX][EY], char *varname)
Read 2-D meteo variable.
- void [read_met_bin_3d](#) (FILE *in, ctl_t *ctl, met_t *met, float var[EX][EY][EP], char *varname, int precision, double tolerance)
Read 3-D meteo variable.
- void [read_met_cape](#) (clim_t *clim, met_t *met)
Calculate convective available potential energy.

- void `read_met_cloud` (`ctl_t *ctl`, `met_t *met`)
Calculate cloud properties.
- void `read_met_detrend` (`ctl_t *ctl`, `met_t *met`)
Apply detrending method to temperature and winds.
- void `read_met_extrapolate` (`met_t *met`)
Extrapolate meteo data at lower boundary.
- void `read_met_geopot` (`ctl_t *ctl`, `met_t *met`)
Calculate geopotential heights.
- void `read_met_grid` (`char *filename`, `int ncid`, `ctl_t *ctl`, `met_t *met`)
Read coordinates of meteo data.
- void `read_met_levels` (`int ncid`, `ctl_t *ctl`, `met_t *met`)
Read meteo data on vertical levels.
- void `read_met_ml2pl` (`ctl_t *ctl`, `met_t *met`, `float var[EX][EY][EP]`)
Convert meteo data from model levels to pressure levels.
- int `read_met_nc_2d` (`int ncid`, `char *varname`, `char *varname2`, `ctl_t *ctl`, `met_t *met`, `float dest[EX][EY]`, `float scl`, `int init`)
Read and convert 2D variable from meteo data file.
- int `read_met_nc_3d` (`int ncid`, `char *varname`, `char *varname2`, `ctl_t *ctl`, `met_t *met`, `float dest[EX][EY][EP]`, `float scl`, `int init`)
Read and convert 3D variable from meteo data file.
- void `read_met_pbl` (`met_t *met`)
Calculate pressure of the boundary layer.
- void `read_met_periodic` (`met_t *met`)
Create meteo data with periodic boundary conditions.
- void `read_met_pv` (`met_t *met`)
Calculate potential vorticity.
- void `read_met_sample` (`ctl_t *ctl`, `met_t *met`)
Downsampling of meteo data.
- void `read_met_surface` (`int ncid`, `met_t *met`, `ctl_t *ctl`)
Read surface data.
- void `read_met_tropo` (`ctl_t *ctl`, `clim_t *clim`, `met_t *met`)
Calculate tropopause data.
- void `read_obs` (`char *filename`, `double *rt`, `double *rz`, `double *rlon`, `double *rlat`, `double *robs`, `int *nobs`)
Read observation data.
- double `scan_ctl` (`const char *filename`, `int argc`, `char *argv[]`, `const char *varname`, `int arridx`, `const char *defvalue`, `char *value`)
Read a control parameter from file or command line.
- double `sedi` (`double p`, `double T`, `double rp`, `double rhop`)
Calculate sedimentation velocity.
- void `spline` (`double *x`, `double *y`, `int n`, `double *x2`, `double *y2`, `int n2`, `int method`)
Spline interpolation.
- float `stddev` (`float *data`, `int n`)
Calculate standard deviation.
- double `sza` (`double sec`, `double lon`, `double lat`)
Calculate solar zenith angle.
- void `time2jsec` (`int year`, `int mon`, `int day`, `int hour`, `int min`, `int sec`, `double remain`, `double *jsec`)
Convert date to seconds.
- void `timer` (`const char *name`, `const char *group`, `int output`)
Measure wall-clock time.
- double `tropo_weight` (`clim_t *clim`, `double t`, `double lat`, `double p`)
Get weighting factor based on tropopause distance.

- void [write_atm](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write atmospheric data.
- void [write_atm_asc](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write atmospheric data in ASCII format.
- void [write_atm_bin](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm)
Write atmospheric data in binary format.
- void [write_atm_clams](#) ([ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write atmospheric data in CLaMS format.
- void [write_atm_nc](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm)
Write atmospheric data in netCDF format.
- void [write_csi](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write CSI data.
- void [write_ens](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write ensemble data.
- void [write_grid](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write gridded data.
- void [write_grid_asc](#) (const char *filename, [ctl_t](#) *ctl, double *cd, double *vmr_expl, double *vmr_impl, double t, double *z, double *lon, double *lat, double *area, double dz, int *np)
Write gridded data in ASCII format.
- void [write_grid_nc](#) (const char *filename, [ctl_t](#) *ctl, double *cd, double *vmr_expl, double *vmr_impl, double t, double *z, double *lon, double *lat, double *area, double dz, int *np)
Write gridded data in netCDF format.
- int [write_met](#) (char *filename, [ctl_t](#) *ctl, [met_t](#) *met)
Read meteo data file.
- void [write_met_bin_2d](#) (FILE *out, [met_t](#) *met, float var[EX][EY], char *varname)
Write 2-D meteo variable.
- void [write_met_bin_3d](#) (FILE *out, [ctl_t](#) *ctl, [met_t](#) *met, float var[EX][EY][EP], char *varname, int precision, double tolerance)
Write 3-D meteo variable.
- void [write_prof](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write profile data.
- void [write_sample](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write sample data.
- void [write_station](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write station data.

5.21.1 Detailed Description

MPTRAC library definitions.

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

5.21.2 Function Documentation

5.21.2.1 buoyancy_frequency() double buoyancy_frequency (

```

    double p0,
    double t0,
    double p1,
    double t1 )

```

Calculate buoyancy frequency.

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

```

00033     {
00034
00035     double theta0 = THETA(p0, t0);
00036     double theta1 = THETA(p1, t1);
00037     double dz = RI / MA / G0 * 0.5 * (t0 + t1) * (log(p0) - log(p1));
00038
00039     return sqrt(2. * G0 / (theta0 + theta1) * (theta1 - theta0) / dz);
00040 }

```

5.21.2.2 cart2geo() void cart2geo (

```

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

```

Convert Cartesian coordinates to geolocation.

Definition at line 44 of file [libtrac.c](#).

```

00048     {
00049
00050     double radius = NORM(x);
00051     *lat = asin(x[2] / radius) * 180. / M_PI;
00052     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00053     *z = radius - RE;
00054 }

```

5.21.2.3 clim_hno3() double clim_hno3 (

```

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

```

Climatology of HNO3 volume mixing ratios.

Definition at line 58 of file [libtrac.c](#).

```

00062     {
00063
00064     /* Get seconds since begin of year... */
00065     double sec = FMOD(t, 365.25 * 86400.);
00066     while (sec < 0)
00067         sec += 365.25 * 86400.;
00068
00069     /* Check pressure... */
00070     if (p < clim->hno3_p[0])
00071         p = clim->hno3_p[0];
00072     else if (p > clim->hno3_p[clim->hno3_np - 1])
00073         p = clim->hno3_p[clim->hno3_np - 1];
00074
00075     /* Check latitude... */
00076     if (lat < clim->hno3_lat[0])
00077         lat = clim->hno3_lat[0];
00078     else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
00079         lat = clim->hno3_lat[clim->hno3_nlat - 1];
00080
00081     /* Get indices... */

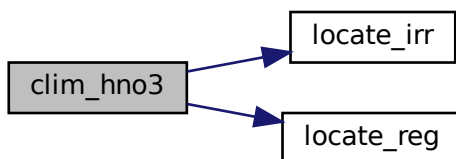
```

```

00082 int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
00083 int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
00084 int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);
00085
00086 /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00087 double aux00 = LIN(clim->hno3_p[ip],
00088                   clim->hno3[isec][ilat][ip],
00089                   clim->hno3_p[ip + 1],
00090                   clim->hno3[isec][ilat][ip + 1], p);
00091 double aux01 = LIN(clim->hno3_p[ip],
00092                   clim->hno3[isec][ilat + 1][ip],
00093                   clim->hno3_p[ip + 1],
00094                   clim->hno3[isec][ilat + 1][ip + 1], p);
00095 double aux10 = LIN(clim->hno3_p[ip],
00096                   clim->hno3[isec + 1][ilat][ip],
00097                   clim->hno3_p[ip + 1],
00098                   clim->hno3[isec + 1][ilat][ip + 1], p);
00099 double aux11 = LIN(clim->hno3_p[ip],
00100                   clim->hno3[isec + 1][ilat + 1][ip],
00101                   clim->hno3_p[ip + 1],
00102                   clim->hno3[isec + 1][ilat + 1][ip + 1], p);
00103 aux00 = LIN(clim->hno3_lat[ilat], aux00,
00104            clim->hno3_lat[ilat + 1], aux01, lat);
00105 aux11 = LIN(clim->hno3_lat[ilat], aux10,
00106            clim->hno3_lat[ilat + 1], aux11, lat);
00107 aux00 = LIN(clim->hno3_time[isec], aux00,
00108            clim->hno3_time[isec + 1], aux11, sec);
00109
00110 /* Convert from ppb to ppv... */
00111 return GSL_MAX(1e-9 * aux00, 0.0);
00112 }

```

Here is the call graph for this function:



5.21.2.4 clim_hno3_init() void clim_hno3_init (
 clim_t * clim)

Initialization function for HNO3 climatology.

Definition at line 116 of file libtrac.c.

```

00117 {
00118
00119 /* Write info... */
00120 LOG(1, "Initialize HNO3 data...");
00121
00122 clim->hno3_ntime = 12;
00123 double hno3_time[12] = {
00124     1209600.00, 3888000.00, 6393600.00,
00125     9072000.00, 11664000.00, 14342400.00,
00126     16934400.00, 19612800.00, 22291200.00,
00127     24883200.00, 27561600.00, 30153600.00
00128 };
00129 memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));
00130
00131 clim->hno3_nlat = 18;
00132 double hno3_lat[18] = {

```

```
00133     -85, -75, -65, -55, -45, -35, -25, -15, -5,
00134     5, 15, 25, 35, 45, 55, 65, 75, 85
00135 };
00136 memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));
00137
00138 clim->hno3_np = 10;
00139 double hno3_p[10] = {
00140     4.64159, 6.81292, 10, 14.678, 21.5443,
00141     31.6228, 46.4159, 68.1292, 100, 146.78
00142 };
00143 memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));
00144
00145 double hno3[12][18][10] = {
00146     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00147      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00148      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00149      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00150      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00151      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00152      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00153      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00154      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00155      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00156      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00157      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00158      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00159      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00160      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00161      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00162      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00163      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00164     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00165      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00166      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00167      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00168      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00169      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00170      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00171      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00172      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00173      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00174      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00175      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00176      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00177      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00178      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00179      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00180      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00181      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00182     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00183      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00184      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00185      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00186      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00187      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00188      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00189      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00190      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00191      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00192      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00193      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00194      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00195      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00196      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00197      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00198      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00199      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00200     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00201      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00202      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00203      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00204      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00205      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00206      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00207      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00208      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00209      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00210      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00211      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00212      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00213      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00214      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00215      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00216      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00217      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00218     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00219      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
```

```
00220 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00221 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00222 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00223 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00224 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00225 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00226 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00227 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00228 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00229 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00230 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00231 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00232 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00233 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00234 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00235 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00236 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00237 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00238 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00239 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00240 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00241 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00242 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00243 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00244 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00245 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00246 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00247 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00248 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00249 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00250 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00251 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00252 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00253 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},
00254 {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
00255 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00256 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00257 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00258 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00259 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00260 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00261 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00262 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00263 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00264 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00265 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00266 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00267 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00268 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00269 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00270 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00271 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62},
00272 {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
00273 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00274 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00275 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00276 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00277 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00278 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00279 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00280 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00281 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00282 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00283 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00284 {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00285 {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00286 {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00287 {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00288 {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00289 {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55},
00290 {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00291 {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00292 {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00293 {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00294 {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00295 {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00296 {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00297 {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00298 {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00299 {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00300 {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00301 {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00302 {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00303 {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00304 {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00305 {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00306 {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
```

```

00307     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00308     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91}},
00309     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84}},
00310     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97}},
00311     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56}},
00312     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11}},
00313     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616}},
00314     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21}},
00315     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968}},
00316     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105}},
00317     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146}},
00318     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178}},
00319     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14}},
00320     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353}},
00321     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802}},
00322     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2}},
00323     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52}},
00324     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73}},
00325     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00326     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78}},
00327     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73}},
00328     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75}},
00329     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41}},
00330     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955}},
00331     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61}},
00332     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269}},
00333     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132}},
00334     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121}},
00335     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147}},
00336     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146}},
00337     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172}},
00338     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448}},
00339     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948}},
00340     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56}},
00341     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76}},
00342     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97}},
00343     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00344     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89}},
00345     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74}},
00346     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65}},
00347     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28}},
00348     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837}},
00349     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488}},
00350     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256}},
00351     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198}},
00352     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173}},
00353     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138}},
00354     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133}},
00355     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189}},
00356     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6}},
00357     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39}},
00358     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85}},
00359     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24}},
00360     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35}},
00361     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00362 };
00363 memcpy(clim->hno3, hno3, sizeof(clim->hno3));
00364
00365 /* Get range... */
00366 double hno3min = 1e99, hno3max = -1e99;
00367 for (int it = 0; it < clim->hno3_ntime; it++)
00368     for (int iz = 0; iz < clim->hno3_np; iz++)
00369         for (int iy = 0; iy < clim->hno3_nlat; iy++) {
00370             hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
00371             hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
00372         }
00373
00374 /* Write info... */
00375 LOG(2, "Number of time steps: %d", clim->hno3_ntime);
00376 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00377     clim->hno3_time[0], clim->hno3_time[1],
00378     clim->hno3_time[clim->hno3_ntime - 1]);
00379 LOG(2, "Number of pressure levels: %d", clim->hno3_np);
00380 LOG(2, "Altitude levels: %g, %g ... %g km",
00381     Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
00382     Z(clim->hno3_p[clim->hno3_np - 1]));
00383 LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
00384     clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
00385 LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
00386 LOG(2, "Latitudes: %g, %g ... %g deg",
00387     clim->hno3_lat[0], clim->hno3_lat[1],
00388     clim->hno3_lat[clim->hno3_nlat - 1]);
00389 LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
00390     1e-9 * hno3max);
00391 }

```

```

5.21.2.5 clim_oh() double clim_oh (
    clim_t * clim,
    double t,
    double lat,
    double p )

```

Climatology of OH number concentrations.

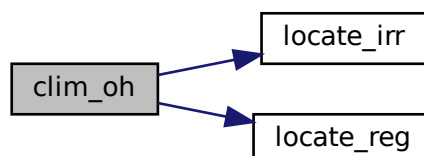
Definition at line 395 of file `libtrac.c`.

```

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

```

Here is the call graph for this function:



5.21.2.6 clim_oh_diurnal() double clim_oh_diurnal (

```

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

```

Climatology of OH number concentrations with diurnal variation.

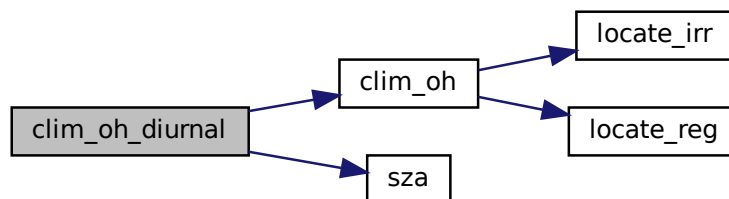
Definition at line 450 of file [libtrac.c](#).

```

00456     {
00457
00458     double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);
00459
00460     if (sza2 <= M_PI / 2. * 89. / 90.)
00461         return oh * exp(-ctl->oh_chem_beta / cos(sza2));
00462     else
00463         return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
00464 }

```

Here is the call graph for this function:



5.21.2.7 clim_oh_init() void clim_oh_init (

```

    ctl_t * ctl,
    clim_t * clim )

```

Initialization function for OH climatology.

Definition at line 468 of file [libtrac.c](#).

```

00470     {
00471
00472     int nt, ncid, varid;
00473
00474     double *help, ohmin = 1e99, ohmax = -1e99;
00475
00476     /* Write info... */
00477     LOG(1, "Read OH data: %s", ctl->clim_oh_filename);
00478
00479     /* Open netCDF file... */
00480     if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00481         WARN("OH climatology data are missing!");
00482         return;
00483     }
00484
00485     /* Read pressure data... */
00486     NC_INQ_DIM("press", &clim->oh_np, 2, CP);
00487     NC_GET_DOUBLE("press", clim->oh_p, 1);
00488
00489     /* Check ordering of pressure data... */

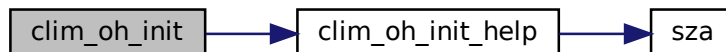
```

```

00490  if (clim->oh_p[0] < clim->oh_p[1])
00491      ERRMSG("Pressure data are not descending!");
00492
00493  /* Read latitudes... */
00494  NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
00495  NC_GET_DOUBLE("lat", clim->oh_lat, 1);
00496
00497  /* Check ordering of latitudes... */
00498  if (clim->oh_lat[0] > clim->oh_lat[1])
00499      ERRMSG("Latitude data are not ascending!");
00500
00501  /* Set time data for monthly means... */
00502  clim->oh_ntime = 12;
00503  clim->oh_time[0] = 1209600.00;
00504  clim->oh_time[1] = 3888000.00;
00505  clim->oh_time[2] = 6393600.00;
00506  clim->oh_time[3] = 9072000.00;
00507  clim->oh_time[4] = 11664000.00;
00508  clim->oh_time[5] = 14342400.00;
00509  clim->oh_time[6] = 16934400.00;
00510  clim->oh_time[7] = 19612800.00;
00511  clim->oh_time[8] = 22291200.00;
00512  clim->oh_time[9] = 24883200.00;
00513  clim->oh_time[10] = 27561600.00;
00514  clim->oh_time[11] = 30153600.00;
00515
00516  /* Check number of timesteps... */
00517  NC_INQ_DIM("time", &nt, 12, 12);
00518
00519  /* Read OH data... */
00520  ALLOC(help, double,
00521        clim->oh_nlat * clim->oh_np * clim->oh_ntime);
00522  NC_GET_DOUBLE("OH", help, 1);
00523  for (int it = 0; it < clim->oh_ntime; it++)
00524      for (int iz = 0; iz < clim->oh_np; iz++)
00525          for (int iy = 0; iy < clim->oh_nlat; iy++) {
00526              clim->oh[it][iz][iy] =
00527                  help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
00528                  / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
00529                                     clim->oh_lat[iy]);
00530              ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
00531              ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
00532          }
00533  free(help);
00534
00535  /* Close netCDF file... */
00536  NC(nc_close(ncid));
00537
00538  /* Write info... */
00539  LOG(2, "Number of time steps: %d", clim->oh_ntime);
00540  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00541      clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
00542  LOG(2, "Number of pressure levels: %d", clim->oh_np);
00543  LOG(2, "Altitude levels: %g, %g ... %g km",
00544      Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
00545  LOG(2, "Pressure levels: %g, %g ... %g hPa",
00546      clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
00547  LOG(2, "Number of latitudes: %d", clim->oh_nlat);
00548  LOG(2, "Latitudes: %g, %g ... %g deg",
00549      clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
00550  LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
00551 }

```

Here is the call graph for this function:



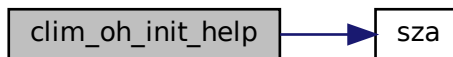
5.21.2.8 clim_oh_init_help() double clim_oh_init_help (
double beta,
double time,
double lat)

Apply diurnal correction to OH climatology.

Definition at line 555 of file libtrac.c.

```
00558     {
00559
00560     double aux, lon, sum = 0;
00561
00562     int n = 0;
00563
00564     /* Integrate day/night correction factor over longitude... */
00565     for (lon = -180; lon < 180; lon += 1) {
00566         aux = sza(time, lon, lat);
00567         if (aux <= M_PI / 2. * 85. / 90.)
00568             sum += exp(-beta / cos(aux));
00569         else
00570             sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
00571         n++;
00572     }
00573     return sum / (double) n;
00574 }
```

Here is the call graph for this function:



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

Climatology of H2O2 number concentrations.

Definition at line 578 of file libtrac.c.

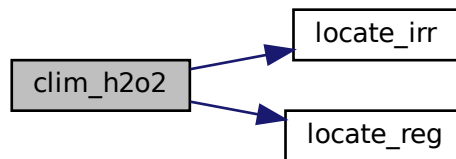
```
00582     {
00583
00584     /* Get seconds since begin of year... */
00585     double sec = FMOD(t, 365.25 * 86400.);
00586     while (sec < 0)
00587         sec += 365.25 * 86400.;
00588
00589     /* Check pressure... */
00590     if (p < clim->h2o2_p[clim->h2o2_np - 1])
00591         p = clim->h2o2_p[clim->h2o2_np - 1];
00592     else if (p > clim->h2o2_p[0])
00593         p = clim->h2o2_p[0];
00594
00595     /* Check latitude... */
00596     if (lat < clim->h2o2_lat[0])
00597         lat = clim->h2o2_lat[0];
00598     else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
00599         lat = clim->h2o2_lat[clim->h2o2_nlat - 1];
00600 }
```

```

00601  /* Get indices... */
00602  int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
00603  int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
00604  int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);
00605
00606  /* Interpolate H2O2 climatology... */
00607  double aux00 = LIN(clim->h2o2_p[ip],
00608                    clim->h2o2[isec][ip][ilat],
00609                    clim->h2o2_p[ip + 1],
00610                    clim->h2o2[isec][ip + 1][ilat], p);
00611  double aux01 = LIN(clim->h2o2_p[ip],
00612                    clim->h2o2[isec][ip][ilat + 1],
00613                    clim->h2o2_p[ip + 1],
00614                    clim->h2o2[isec][ip + 1][ilat + 1], p);
00615  double aux10 = LIN(clim->h2o2_p[ip],
00616                    clim->h2o2[isec + 1][ip][ilat],
00617                    clim->h2o2_p[ip + 1],
00618                    clim->h2o2[isec + 1][ip + 1][ilat], p);
00619  double aux11 = LIN(clim->h2o2_p[ip],
00620                    clim->h2o2[isec + 1][ip][ilat + 1],
00621                    clim->h2o2_p[ip + 1],
00622                    clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
00623  aux00 =
00624      LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
00625  aux11 =
00626      LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
00627  aux00 =
00628      LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);
00629
00630  return GSL_MAX(aux00, 0.0);
00631 }

```

Here is the call graph for this function:



5.21.2.10 clim_h2o2_init() void clim_h2o2_init (

```

    ctl_t * ctl,
    clim_t * clim )

```

Initialization function for H2O2 climatology.

Definition at line 635 of file libtrac.c.

```

00637  {
00638
00639  int ncid, varid, it, iy, iz, nt;
00640
00641  double *help, h2o2min = 1e99, h2o2max = -1e99;
00642
00643  /* Write info... */
00644  LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);
00645
00646  /* Open netCDF file... */
00647  if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00648      WARN("H2O2 climatology data are missing!");
00649      return;
00650  }
00651

```

```

00652  /* Read pressure data... */
00653  NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
00654  NC_GET_DOUBLE("press", clim->h2o2_p, 1);
00655
00656  /* Check ordering of pressure data... */
00657  if (clim->h2o2_p[0] < clim->h2o2_p[1])
00658      ERRMSG("Pressure data are not descending!");
00659
00660  /* Read latitudes... */
00661  NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
00662  NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);
00663
00664  /* Check ordering of latitude data... */
00665  if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
00666      ERRMSG("Latitude data are not ascending!");
00667
00668  /* Set time data (for monthly means)... */
00669  clim->h2o2_ntime = 12;
00670  clim->h2o2_time[0] = 1209600.00;
00671  clim->h2o2_time[1] = 3888000.00;
00672  clim->h2o2_time[2] = 6393600.00;
00673  clim->h2o2_time[3] = 9072000.00;
00674  clim->h2o2_time[4] = 11664000.00;
00675  clim->h2o2_time[5] = 14342400.00;
00676  clim->h2o2_time[6] = 16934400.00;
00677  clim->h2o2_time[7] = 19612800.00;
00678  clim->h2o2_time[8] = 22291200.00;
00679  clim->h2o2_time[9] = 24883200.00;
00680  clim->h2o2_time[10] = 27561600.00;
00681  clim->h2o2_time[11] = 30153600.00;
00682
00683  /* Check number of timesteps... */
00684  NC_INQ_DIM("time", &nt, 12, 12);
00685
00686  /* Read data... */
00687  ALLOC(help, double,
00688         clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
00689  NC_GET_DOUBLE("h2o2", help, 1);
00690  for (it = 0; it < clim->h2o2_ntime; it++)
00691      for (iz = 0; iz < clim->h2o2_np; iz++)
00692          for (iy = 0; iy < clim->h2o2_nlat; iy++) {
00693              clim->h2o2[it][iz][iy] =
00694                  help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
00695              h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
00696              h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
00697          }
00698  free(help);
00699
00700  /* Close netCDF file... */
00701  NC(nc_close(ncid));
00702
00703  /* Write info... */
00704  LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
00705  LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00706       clim->h2o2_time[0], clim->h2o2_time[1],
00707       clim->h2o2_time[clim->h2o2_ntime - 1]);
00708  LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
00709  LOG(2, "Altitude levels: %g, %g ... %g km",
00710       Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
00711       Z(clim->h2o2_p[clim->h2o2_np - 1]));
00712  LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
00713       clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
00714  LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
00715  LOG(2, "Latitudes: %g, %g ... %g deg",
00716       clim->h2o2_lat[0], clim->h2o2_lat[1],
00717       clim->h2o2_lat[clim->h2o2_nlat - 1]);
00718  LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
00719 }

```

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

Climatology of tropopause pressure.

Definition at line 723 of file libtrac.c.

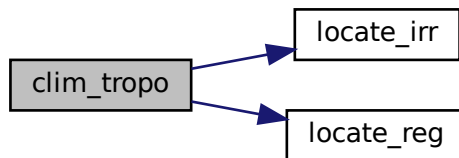
```
00726 {
```

```

00727
00728  /* Get seconds since begin of year... */
00729  double sec = FMOD(t, 365.25 * 86400.);
00730  while (sec < 0)
00731      sec += 365.25 * 86400.;
00732
00733  /* Get indices... */
00734  int isec = locate_irr(clim->tropo_time, clim->tropo_ntime, sec);
00735  int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);
00736
00737  /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
00738  double p0 = LIN(clim->tropo_lat[ilat],
00739                 clim->tropo[isec][ilat],
00740                 clim->tropo_lat[ilat + 1],
00741                 clim->tropo[isec][ilat + 1], lat);
00742  double p1 = LIN(clim->tropo_lat[ilat],
00743                 clim->tropo[isec + 1][ilat],
00744                 clim->tropo_lat[ilat + 1],
00745                 clim->tropo[isec + 1][ilat + 1], lat);
00746  return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
00747 }

```

Here is the call graph for this function:



5.21.2.12 clim_tropo_init() void clim_tropo_init (
 clim_t * clim)

Initialize tropopause climatology.

Definition at line 751 of file libtrac.c.

```

00752  {
00753
00754  /* Write info... */
00755  LOG(1, "Initialize tropopause data...");
00756
00757  clim->tropo_ntime = 12;
00758  double tropo_time[12] = {
00759      1209600.00, 3888000.00, 6393600.00,
00760      9072000.00, 11664000.00, 14342400.00,
00761      16934400.00, 19612800.00, 22291200.00,
00762      24883200.00, 27561600.00, 30153600.00
00763  };
00764  memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));
00765
00766  clim->tropo_nlat = 73;
00767  double tropo_lat[73] = {
00768      -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00769      -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00770      -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00771      -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00772      15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00773      45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00774      75, 77.5, 80, 82.5, 85, 87.5, 90
00775  };
00776  memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));
00777

```

```
00778 double tropo[12][73] = {
00779     {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00780      297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00781      175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00782      99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00783      98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00784      152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00785      277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00786      275.3, 275.6, 275.4, 274.1, 273.5},
00787     {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00788      300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00789      150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00790      98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00791      98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00792      220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00793      284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00794      287.5, 286.2, 285.8},
00795     {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00796      297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00797      161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00798      100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00799      99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00800      186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00801      279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00802      304.3, 304.9, 306, 306.6, 306.2, 306},
00803     {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00804      290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00805      195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00806      102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00807      99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00808      148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00809      263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00810      315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00811     {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00812      260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00813      205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00814      101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00815      102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00816      165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00817      273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00818      325.3, 325.8, 325.8},
00819     {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00820      222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00821      228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00822      105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00823      106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00824      127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00825      251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00826      308.5, 312.2, 313.1, 313.3},
00827     {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00828      187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00829      235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00830      110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00831      111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00832      117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00833      224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00834      275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00835     {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00836      185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00837      233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00838      110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00839      112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00840      120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00841      230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00842      278.2, 282.6, 287.4, 290.9, 292.5, 293},
00843     {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00844      183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00845      243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00846      114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00847      110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00848      114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00849      203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00850      276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00851     {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00852      215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00853      237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00854      111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00855      106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00856      112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00857      206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00858      279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00859      305.1},
00860     {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00861      253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00862      223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00863      108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00864      102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
```

```

00865     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00866     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00867     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00868     {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00869     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00870     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00871     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00872     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00873     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00874     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00875     281.7, 281.1, 281.2}
00876 };
00877 memcpy(clim->tropo, tropo, sizeof(clim->tropo));
00878
00879 /* Get range... */
00880 double tropomin = 1e99, tropomax = -1e99;
00881 for (int it = 0; it < clim->tropo_ntime; it++)
00882     for (int iy = 0; iy < clim->tropo_nlat; iy++) {
00883         tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
00884         tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
00885     }
00886
00887 /* Write info... */
00888 LOG(2, "Number of time steps: %d", clim->tropo_ntime);
00889 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00890     clim->tropo_time[0], clim->tropo_time[1],
00891     clim->tropo_time[clim->tropo_ntime - 1]);
00892 LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
00893 LOG(2, "Latitudes: %g, %g ... %g deg",
00894     clim->tropo_lat[0], clim->tropo_lat[1],
00895     clim->tropo_lat[clim->tropo_nlat - 1]);
00896 LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
00897     Z(tropomin));
00898 LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
00899 }

```

5.21.2.13 compress_pack() void compress_pack (

```

    char * varname,
    float * array,
    size_t nxy,
    size_t nz,
    int decompress,
    FILE * inout )

```

Pack or unpack array.

Definition at line 903 of file libtrac.c.

```

00909     {
00910
00911     double min[EP], max[EP], off[EP], scl[EP];
00912
00913     unsigned short *sarray;
00914
00915     /* Allocate... */
00916     ALLOC(sarray, unsigned short,
00917         nxy * nz);
00918
00919     /* Read compressed stream and decompress array... */
00920     if (decompress) {
00921
00922         /* Write info... */
00923         LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
00924             varname, 100. * sizeof(unsigned short) / sizeof(float));
00925
00926         /* Read data... */
00927         FREAD(&scl, double,
00928             nz,
00929             inout);
00930         FREAD(&off, double,
00931             nz,
00932             inout);
00933         FREAD(sarray, unsigned short,
00934             nxy * nz,
00935             inout);
00936
00937         /* Convert to float... */

```



```

00938 #pragma omp parallel for default(shared)
00939     for (size_t ixy = 0; ixy < nxy; ixy++)
00940         for (size_t iz = 0; iz < nz; iz++)
00941             array[ixy * nz + iz]
00942                 = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
00943 }
00944
00945 /* Compress array and output compressed stream... */
00946 else {
00947
00948     /* Write info... */
00949     LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
00950         varname, 100. * sizeof(unsigned short) / sizeof(float));
00951
00952     /* Get range... */
00953     for (size_t iz = 0; iz < nz; iz++) {
00954         min[iz] = array[iz];
00955         max[iz] = array[iz];
00956     }
00957     for (size_t ixy = 1; ixy < nxy; ixy++)
00958         for (size_t iz = 0; iz < nz; iz++) {
00959             if (array[ixy * nz + iz] < min[iz])
00960                 min[iz] = array[ixy * nz + iz];
00961             if (array[ixy * nz + iz] > max[iz])
00962                 max[iz] = array[ixy * nz + iz];
00963         }
00964
00965     /* Get offset and scaling factor... */
00966     for (size_t iz = 0; iz < nz; iz++) {
00967         scl[iz] = (max[iz] - min[iz]) / 65533.;
00968         off[iz] = min[iz];
00969     }
00970
00971     /* Convert to short... */
00972 #pragma omp parallel for default(shared)
00973     for (size_t ixy = 0; ixy < nxy; ixy++)
00974         for (size_t iz = 0; iz < nz; iz++)
00975             if (scl[iz] != 0)
00976                 sarray[ixy * nz + iz] = (unsigned short)
00977                     ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
00978             else
00979                 sarray[ixy * nz + iz] = 0;
00980
00981     /* Write data... */
00982     FWRITE(&scl, double,
00983         nz,
00984         inout);
00985     FWRITE(&off, double,
00986         nz,
00987         inout);
00988     FWRITE(sarray, unsigned short,
00989         nxy * nz,
00990         inout);
00991 }
00992
00993 /* Free... */
00994 free(sarray);
00995 }

```

5.21.2.14 day2doy() void day2doy (

```

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

```

Compress or decompress array with zfp.

Compress or decompress array with zstd.

Get day of year from date.

Definition at line 1144 of file libtrac.c.

```

01148 {
01149
01150     const int
01151     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },

```

```
01152     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01153
01154     /* Get day of year... */
01155     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01156         *doy = d0l[mon - 1] + day - 1;
01157     else
01158         *doy = d0[mon - 1] + day - 1;
01159 }
```

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

Get date from day of year.

Definition at line 1163 of file libtrac.c.

```
01167     {
01168
01169     const int
01170     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01171     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01172
01173     int i;
01174
01175     /* Get month and day... */
01176     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01177         for (i = 11; i > 0; i--)
01178             if (d0l[i] <= doy)
01179                 break;
01180         *mon = i + 1;
01181         *day = doy - d0l[i] + 1;
01182     } else {
01183         for (i = 11; i > 0; i--)
01184             if (d0[i] <= doy)
01185                 break;
01186         *mon = i + 1;
01187         *day = doy - d0[i] + 1;
01188     }
01189 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1193 of file libtrac.c.

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

5.21.2.17 get_met() void get_met (

```

    ctl_t * ctl,
    clim_t * clim,
    double t,
    met_t ** met0,
    met_t ** met1 )

```

Get meteo data for given time step.

Definition at line 1207 of file libtrac.c.

```

01212     {
01213
01214     static int init;
01215
01216     met_t *mets;
01217
01218     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01219
01220     /* Set timer... */
01221     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01222
01223     /* Init... */
01224     if (t == ctl->t_start || !init) {
01225         init = 1;
01226
01227         /* Read meteo data... */
01228         get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
01229                     ctl->metbase, ctl->dt_met, filename);
01230         if (!read_met(filename, ctl, clim, *met0))
01231             ERRMSG("Cannot open file!");
01232
01233         get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
01234                     ctl->metbase, ctl->dt_met, filename);
01235         if (!read_met(filename, ctl, clim, *met1))
01236             ERRMSG("Cannot open file!");
01237
01238         /* Update GPU... */
01239         #ifdef _OPENACC
01240             met_t *met0up = *met0;
01241             met_t *met1up = *met1;
01242         #endif
01243         #pragma acc update device(met0up[:1],met1up[:1]) async(5)
01244         #else
01245             #pragma acc update device(met0up[:1],met1up[:1])
01246         #endif
01247         #endif
01248
01249         /* Caching... */
01250         if (ctl->met_cache && t != ctl->t_stop) {
01251             get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
01252                         ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
01253             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01254             LOG(1, "Caching: %s", cachefile);
01255             if (system(cmd) != 0)
01256                 WARN("Caching command failed!");
01257         }
01258     }
01259
01260     /* Read new data for forward trajectories... */
01261     if (t > (*met1)->time) {
01262
01263         /* Pointer swap... */
01264         mets = *met1;
01265         *met1 = *met0;
01266         *met0 = mets;
01267
01268         /* Read new meteo data... */
01269         get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
01270         if (!read_met(filename, ctl, clim, *met1))
01271             ERRMSG("Cannot open file!");
01272
01273         /* Update GPU... */
01274         #ifdef _OPENACC
01275             met_t *met1up = *met1;
01276         #endif
01277         #pragma acc update device(met1up[:1]) async(5)
01278         #else
01279             #pragma acc update device(met1up[:1])
01280         #endif
01281         #endif
01282
01283         /* Caching... */
01284         if (ctl->met_cache && t != ctl->t_stop) {

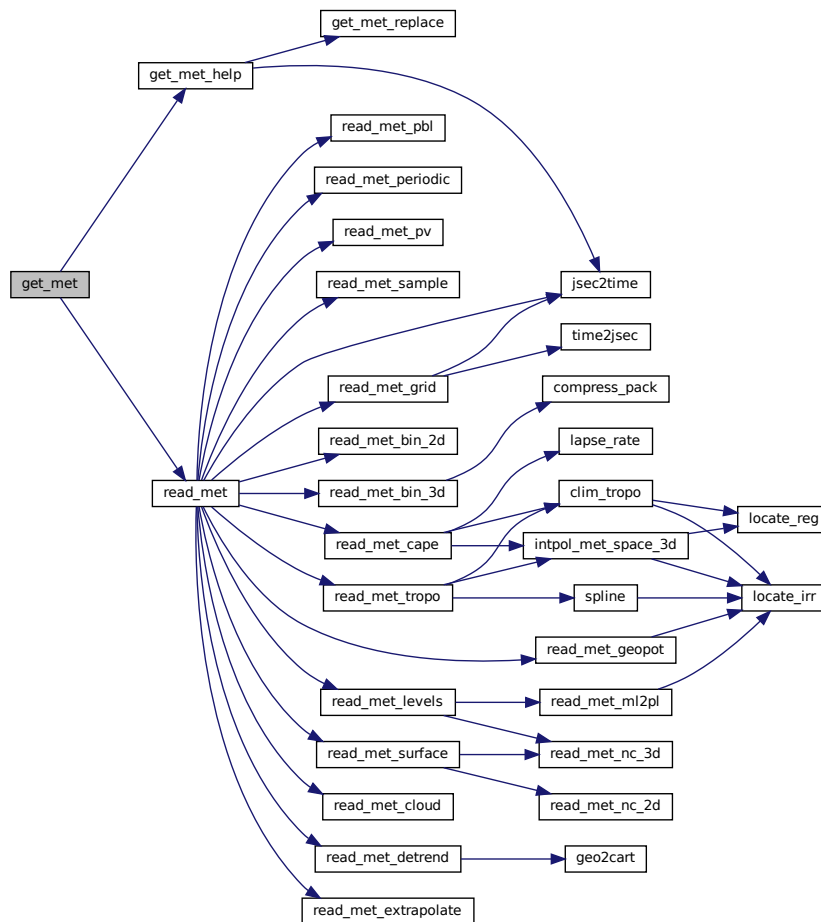
```

```

01285     get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
01286                  cachefile);
01287     sprintf(cmd, "cat %s > /dev/null &", cachefile);
01288     LOG(1, "Caching: %s", cachefile);
01289     if (system(cmd) != 0)
01290         WARN("Caching command failed!");
01291 }
01292 }
01293 /* Read new data for backward trajectories... */
01294 if (t < (*met0)->time) {
01295
01296     /* Pointer swap... */
01297     mets = *met1;
01298     *met1 = *met0;
01299     *met0 = mets;
01300
01301     /* Read new meteo data... */
01302     get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
01303     if (!read_met(filename, ctl, clim, *met0))
01304         ERRMSG("Cannot open file!");
01305
01306     /* Update GPU... */
01307 #ifdef _OPENACC
01308     met_t *met0up = *met0;
01309 #ifdef ASYNCIO
01310 #pragma acc update device(met0up[:1]) async(5)
01311 #else
01312 #pragma acc update device(met0up[:1])
01313 #endif
01314 #endif
01315
01316     /* Caching... */
01317     if (ctl->met_cache && t != ctl->t_stop) {
01318         get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
01319                     cachefile);
01320         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01321         LOG(1, "Caching: %s", cachefile);
01322         if (system(cmd) != 0)
01323             WARN("Caching command failed!");
01324     }
01325 }
01326 /* Check that grids are consistent... */
01327 if ((*met0)->nx != 0 && (*met1)->nx != 0) {
01328     if ((*met0)->nx != (*met1)->nx
01329         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01330         ERRMSG("Meteo grid dimensions do not match!");
01331     for (int ix = 0; ix < (*met0)->nx; ix++)
01332         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01333             ERRMSG("Meteo grid longitudes do not match!");
01334     for (int iy = 0; iy < (*met0)->ny; iy++)
01335         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01336             ERRMSG("Meteo grid latitudes do not match!");
01337     for (int ip = 0; ip < (*met0)->np; ip++)
01338         if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01339             ERRMSG("Meteo grid pressure levels do not match!");
01340 }
01341 }

```

Here is the call graph for this function:



5.21.2.18 get_met_help() void get_met_help (

```

    ctl_t * ctl,
    double t,
    int direct,
    char * metbase,
    double dt_met,
    char * filename )

```

Get meteo data for time step.

Definition at line 1345 of file libtrac.c.

```

01351 {
01352
01353     char repl[LEN];
01354
01355     double t6, r;
01356
01357     int year, mon, day, hour, min, sec;
01358
01359     /* Round time to fixed intervals... */
01360     if (direct == -1)
01361         t6 = floor(t / dt_met) * dt_met;

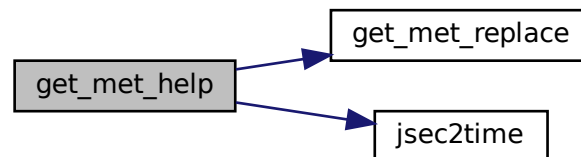
```

```

01362     else
01363         t6 = ceil(t / dt_met) * dt_met;
01364
01365     /* Decode time... */
01366     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01367
01368     /* Set filename of MPTRAC meteo files... */
01369     if (ctl->clams_met_data == 0) {
01370         if (ctl->met_type == 0)
01371             sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01372         else if (ctl->met_type == 1)
01373             sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
01374         else if (ctl->met_type == 2)
01375             sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
01376         else if (ctl->met_type == 3)
01377             sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
01378         else if (ctl->met_type == 4)
01379             sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", metbase);
01380         sprintf(repl, "%d", year);
01381         get_met_replace(filename, "YYYY", repl);
01382         sprintf(repl, "%02d", mon);
01383         get_met_replace(filename, "MM", repl);
01384         sprintf(repl, "%02d", day);
01385         get_met_replace(filename, "DD", repl);
01386         sprintf(repl, "%02d", hour);
01387         get_met_replace(filename, "HH", repl);
01388     }
01389
01390     /* Set filename of CLaMS meteo files... */
01391     else {
01392         sprintf(filename, "%s_YYMMDDHH.nc", metbase);
01393         sprintf(repl, "%d", year);
01394         get_met_replace(filename, "YYYY", repl);
01395         sprintf(repl, "%d", year % 100);
01396         get_met_replace(filename, "YY", repl);
01397         sprintf(repl, "%02d", mon);
01398         get_met_replace(filename, "MM", repl);
01399         sprintf(repl, "%02d", day);
01400         get_met_replace(filename, "DD", repl);
01401         sprintf(repl, "%02d", hour);
01402         get_met_replace(filename, "HH", repl);
01403     }
01404 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1408 of file libtrac.c.

```

01411     {

```

```

01412
01413     char buffer[LEN];
01414
01415     /* Iterate... */
01416     for (int i = 0; i < 3; i++) {
01417
01418         /* Replace sub-string... */
01419         char *ch;
01420         if (!(ch = strstr(orig, search)))
01421             return;
01422         strncpy(buffer, orig, (size_t) (ch - orig));
01423         buffer[ch - orig] = 0;
01424         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01425         orig[0] = 0;
01426         strcpy(orig, buffer);
01427     }
01428 }

```

5.21.2.20 intpol_met_space_3d() void intpol_met_space_3d (

```

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

```

Spatial interpolation of meteo data.

Definition at line 1432 of file libtrac.c.

```

01441     {
01442
01443     /* Initialize interpolation... */
01444     if (init) {
01445
01446         /* Check longitude... */
01447         if (met->lon[met->nx - 1] > 180 && lon < 0)
01448             lon += 360;
01449
01450         /* Get interpolation indices... */
01451         ci[0] = locate_irr(met->p, met->np, p);
01452         ci[1] = locate_reg(met->lon, met->nx, lon);
01453         ci[2] = locate_reg(met->lat, met->ny, lat);
01454
01455         /* Get interpolation weights... */
01456         cw[0] = (met->p[ci[0] + 1] - p)
01457             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01458         cw[1] = (met->lon[ci[1] + 1] - lon)
01459             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01460         cw[2] = (met->lat[ci[2] + 1] - lat)
01461             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01462     }
01463
01464     /* Interpolate vertically... */
01465     double aux00 =
01466         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01467         + array[ci[1]][ci[2]][ci[0] + 1];
01468     double aux01 =
01469         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01470             array[ci[1]][ci[2] + 1][ci[0] + 1])
01471         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01472     double aux10 =
01473         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01474             array[ci[1] + 1][ci[2]][ci[0] + 1])
01475         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01476     double aux11 =
01477         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01478             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01479         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01480
01481     /* Interpolate horizontally... */
01482     aux00 = cw[2] * (aux00 - aux01) + aux01;
01483     aux11 = cw[2] * (aux10 - aux11) + aux11;

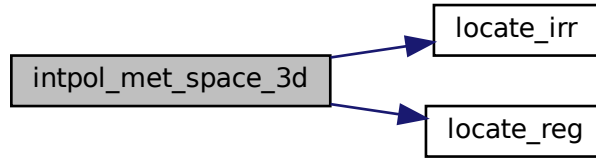
```

```

01484  *var = cw[1] * (aux00 - aux11) + aux11;
01485 }

```

Here is the call graph for this function:



5.21.2.21 intpol_met_space_2d() void intpol_met_space_2d (

```

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

```

Spatial interpolation of meteo data.

Definition at line 1489 of file libtrac.c.

```

01497     {
01498
01499     /* Initialize interpolation... */
01500     if (init) {
01501
01502         /* Check longitude... */
01503         if (met->lon[met->nx - 1] > 180 && lon < 0)
01504             lon += 360;
01505
01506         /* Get interpolation indices... */
01507         ci[1] = locate_reg(met->lon, met->nx, lon);
01508         ci[2] = locate_reg(met->lat, met->ny, lat);
01509
01510         /* Get interpolation weights... */
01511         cw[1] = (met->lon[ci[1] + 1] - lon)
01512             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01513         cw[2] = (met->lat[ci[2] + 1] - lat)
01514             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01515     }
01516
01517     /* Set variables... */
01518     double aux00 = array[ci[1]][ci[2]];
01519     double aux01 = array[ci[1]][ci[2] + 1];
01520     double aux10 = array[ci[1] + 1][ci[2]];
01521     double aux11 = array[ci[1] + 1][ci[2] + 1];
01522
01523     /* Interpolate horizontally... */
01524     if (isfinite(aux00) && isfinite(aux01)
01525         && isfinite(aux10) && isfinite(aux11)) {
01526         aux00 = cw[2] * (aux00 - aux01) + aux01;
01527         aux11 = cw[2] * (aux10 - aux11) + aux11;
01528         *var = cw[1] * (aux00 - aux11) + aux11;
01529     } else {
01530         if (cw[2] < 0.5) {
01531             if (cw[1] < 0.5)

```

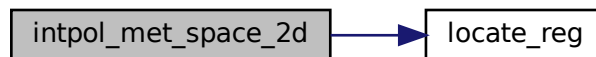


```

01532     *var = aux11;
01533     else
01534     *var = aux01;
01535 } else {
01536     if (cw[1] < 0.5)
01537     *var = aux10;
01538     else
01539     *var = aux00;
01540 }
01541 }
01542 }

```

Here is the call graph for this function:



5.21.2.22 intpol_met_time_3d() void intpol_met_time_3d (

```

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

```

Spatial interpolation of meteo data.

Temporal interpolation of meteo data.

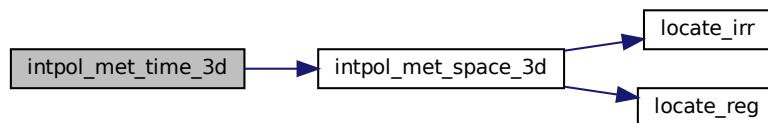
Definition at line 1649 of file libtrac.c.

```

01661     {
01662
01663     double var0, var1, wt;
01664
01665     /* Spatial interpolation... */
01666     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01667     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01668
01669     /* Get weighting factor... */
01670     wt = (met1->time - ts) / (met1->time - met0->time);
01671
01672     /* Interpolate... */
01673     *var = wt * (var0 - var1) + var1;
01674 }

```

Here is the call graph for this function:



5.21.2.23 intpol_met_time_2d() void intpol_met_time_2d (

```

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

```

Temporal interpolation of meteo data.

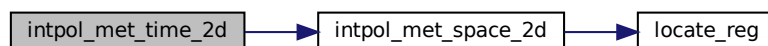
Definition at line 1678 of file libtrac.c.

```

01689     {
01690
01691     double var0, var1, wt;
01692
01693     /* Spatial interpolation... */
01694     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01695     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01696
01697     /* Get weighting factor... */
01698     wt = (met1->time - ts) / (met1->time - met0->time);
01699
01700     /* Interpolate... */
01701     if (isfinite(var0) && isfinite(var1))
01702         *var = wt * (var0 - var1) + var1;
01703     else if (wt < 0.5)
01704         *var = var1;
01705     else
01706         *var = var0;
01707 }

```

Here is the call graph for this function:



5.21.2.24 jsec2time() void jsec2time (

```

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

```

Temporal interpolation of meteo data.

Convert seconds to date.

Definition at line 1742 of file libtrac.c.

```

01750     {
01751
01752     struct tm t0, *t1;
01753
01754     t0.tm_year = 100;
01755     t0.tm_mon = 0;
01756     t0.tm_mday = 1;
01757     t0.tm_hour = 0;
01758     t0.tm_min = 0;
01759     t0.tm_sec = 0;
01760
01761     time_t jsec0 = (time_t) jsec + timegm(&t0);
01762     t1 = gmtime(&jsec0);
01763
01764     *year = t1->tm_year + 1900;
01765     *mon = t1->tm_mon + 1;
01766     *day = t1->tm_mday;
01767     *hour = t1->tm_hour;
01768     *min = t1->tm_min;
01769     *sec = t1->tm_sec;
01770     *remain = jsec - floor(jsec);
01771 }

```

5.21.2.25 lapse_rate() double lapse_rate (

```

    double t,
    double h2o )

```

Calculate moist adiabatic lapse rate.

Definition at line 1775 of file libtrac.c.

```

01777     {
01778
01779     /*
01780      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01781      and water vapor volume mixing ratio [1].
01782
01783      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01784     */
01785
01786     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01787
01788     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01789 }

```

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

Find array index for irregular grid.

Definition at line 1793 of file libtrac.c.

```
01796     {
01797
01798     int ilo = 0;
01799     int ihi = n - 1;
01800     int i = (ihi + ilo) » 1;
01801
01802     if (xx[i] < xx[i + 1])
01803         while (ihi > ilo + 1) {
01804             i = (ihi + ilo) » 1;
01805             if (xx[i] > x)
01806                 ihi = i;
01807             else
01808                 ilo = i;
01809         } else
01810             while (ihi > ilo + 1) {
01811                 i = (ihi + ilo) » 1;
01812                 if (xx[i] <= x)
01813                     ihi = i;
01814                 else
01815                     ilo = i;
01816             }
01817
01818     return ilo;
01819 }
```

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

Find array index for regular grid.

Definition at line 1823 of file libtrac.c.

```
01826     {
01827
01828     /* Calculate index... */
01829     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01830
01831     /* Check range... */
01832     if (i < 0)
01833         return 0;
01834     else if (i > n - 2)
01835         return n - 2;
01836     else
01837         return i;
01838 }
```

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

Calculate NAT existence temperature.

Definition at line 1842 of file libtrac.c.

```
01845     {
01846
01847     /* Check water vapor vmr... */
```

```

01848     h2o = GSL_MAX(h2o, 0.1e-6);
01849
01850     /* Calculate T_NAT... */
01851     double p_hno3 = hno3 * p / 1.333224;
01852     double p_h2o = h2o * p / 1.333224;
01853     double a = 0.009179 - 0.00088 * log10(p_h2o);
01854     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01855     double c = -11397.0 / a;
01856     double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
01857     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01858     if (x2 > 0)
01859         tnat = x2;
01860
01861     return tnat;
01862 }

```

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

Parallel quicksort.

Definition at line 1866 of file libtrac.c.

```

01870     {
01871
01872     if (low < high) {
01873         int pi = quicksort_partition(arr, brr, low, high);
01874
01875     #pragma omp task firstprivate(arr,brr,low,pi)
01876         {
01877             quicksort(arr, brr, low, pi - 1);
01878         }
01879
01880     // #pragma omp task firstprivate(arr,brr,high,pi)
01881         {
01882             quicksort(arr, brr, pi + 1, high);
01883         }
01884     }
01885 }

```

Here is the call graph for this function:



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

Partition function for quicksort.

Definition at line 1889 of file libtrac.c.

```

01893     {
01894
01895     double pivot = arr[high];
01896     int i = (low - 1);
01897
01898     for (int j = low; j <= high - 1; j++)
01899         if (arr[j] <= pivot) {
01900             i++;
01901             SWAP(arr[i], arr[j], double);
01902             SWAP(brr[i], brr[j], int);
01903         }
01904     SWAP(arr[high], arr[i + 1], double);
01905     SWAP(brr[high], brr[i + 1], int);
01906
01907     return (i + 1);
01908 }

```

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

Read atmospheric data.

Definition at line 1912 of file libtrac.c.

```

01915     {
01916
01917     int result;
01918
01919     /* Set timer... */
01920     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
01921
01922     /* Init... */
01923     atm->np = 0;
01924
01925     /* Write info... */
01926     LOG(1, "Read atmospheric data: %s", filename);
01927
01928     /* Read ASCII data... */
01929     if (ctl->atm_type == 0)
01930         result = read_atm_asc(filename, ctl, atm);
01931
01932     /* Read binary data... */
01933     else if (ctl->atm_type == 1)
01934         result = read_atm_bin(filename, ctl, atm);
01935
01936     /* Read netCDF data... */
01937     else if (ctl->atm_type == 2)
01938         result = read_atm_nc(filename, ctl, atm);
01939
01940     /* Read CLaMS data... */
01941     else if (ctl->atm_type == 3)
01942         result = read_atm_clams(filename, ctl, atm);
01943
01944     /* Error... */
01945     else
01946         ERRMSG("Atmospheric data type not supported!");
01947
01948     /* Check result... */
01949     if (result != 1)
01950         return 0;
01951
01952     /* Check number of air parcels... */
01953     if (atm->np < 1)
01954         ERRMSG("Can not read any data!");
01955
01956     /* Write info... */
01957     double mini, maxi;
01958     LOG(2, "Number of particles: %d", atm->np);
01959     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
01960     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
01961     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
01962     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
01963     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
01964     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
01965     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
01966     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);

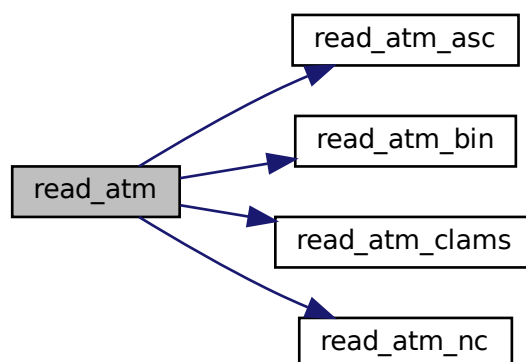
```

```

01967 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
01968 for (int iq = 0; iq < ctl->nq; iq++) {
01969     char msg[LEN];
01970     sprintf(msg, "Quantity %s range: %s ... %s %s",
01971             ctl->qnt_name[iq], ctl->qnt_format[iq],
01972             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
01973     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
01974     LOG(2, msg, mini, maxi);
01975 }
01976
01977 /* Return success... */
01978 return 1;
01979 }

```

Here is the call graph for this function:



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

Read atmospheric data in ASCII format.

Definition at line 1983 of file `libtrac.c`.

```

01986     {
01987
01988     FILE *in;
01989
01990     /* Open file... */
01991     if (!(in = fopen(filename, "r"))) {
01992         WARN("Cannot open file!");
01993         return 0;
01994     }
01995
01996     /* Read line... */
01997     char line[LEN];
01998     while (fgets(line, LEN, in)) {
01999
02000         /* Read data... */
02001         char *tok;
02002         TOK(line, tok, "%lg", atm->time[atm->np]);
02003         TOK(NULL, tok, "%lg", atm->p[atm->np]);
02004         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02005         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02006         for (int iq = 0; iq < ctl->nq; iq++)
02007             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

```

```

02008
02009     /* Convert altitude to pressure... */
02010     atm->p[atm->np] = P(atm->p[atm->np]);
02011
02012     /* Increment data point counter... */
02013     if ((++atm->np) > NP)
02014         ERRMSG("Too many data points!");
02015 }
02016
02017 /* Close file... */
02018 fclose(in);
02019
02020 /* Return success... */
02021 return 1;
02022 }

```

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

Read atmospheric data in binary format.

Definition at line 2026 of file libtrac.c.

```

02029     {
02030
02031     FILE *in;
02032
02033     /* Open file... */
02034     if (!(in = fopen(filename, "r")))
02035         return 0;
02036
02037     /* Check version of binary data... */
02038     int version;
02039     FREAD(&version, int,
02040          1,
02041          in);
02042     if (version != 100)
02043         ERRMSG("Wrong version of binary data!");
02044
02045     /* Read data... */
02046     FREAD(&atm->np, int,
02047          1,
02048          in);
02049     FREAD(atm->time, double,
02050          (size_t) atm->np,
02051          in);
02052     FREAD(atm->p, double,
02053          (size_t) atm->np,
02054          in);
02055     FREAD(atm->lon, double,
02056          (size_t) atm->np,
02057          in);
02058     FREAD(atm->lat, double,
02059          (size_t) atm->np,
02060          in);
02061     for (int iq = 0; iq < ctl->nq; iq++)
02062         FREAD(atm->q[iq], double,
02063              (size_t) atm->np,
02064              in);
02065
02066     /* Read final flag... */
02067     int final;
02068     FREAD(&final, int,
02069          1,
02070          in);
02071     if (final != 999)
02072         ERRMSG("Error while reading binary data!");
02073
02074     /* Close file... */
02075     fclose(in);
02076
02077     /* Return success... */
02078     return 1;
02079 }

```


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

Read atmospheric data in CLaMS format.

Definition at line 2083 of file [libtrac.c](#).

```
02086     {
02087
02088     int ncid, varid;
02089
02090     /* Open file... */
02091     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02092         return 0;
02093
02094     /* Get dimensions... */
02095     NC_INQ_DIM("NPARTS", &atm->np, 1, NP);
02096
02097     /* Get time... */
02098     if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
02099         NC(nc_get_var_double(ncid, varid, atm->time));
02100     } else {
02101         WARN("TIME_INIT not found use time instead!");
02102         double time_init;
02103         NC_GET_DOUBLE("time", &time_init, 1);
02104         for (int ip = 0; ip < atm->np; ip++) {
02105             atm->time[ip] = time_init;
02106         }
02107     }
02108
02109     /* Read zeta coordinate, pressure is optional... */
02110     if (ctl->vert_coord_ap == 1) {
02111         NC_GET_DOUBLE("ZETA", atm->zeta, 1);
02112         NC_GET_DOUBLE("PRESS", atm->p, 0);
02113     }
02114
02115     /* Read pressure, zeta coordinate is optional... */
02116     else {
02117         NC_GET_DOUBLE("PRESS", atm->p, 1);
02118         NC_GET_DOUBLE("ZETA", atm->zeta, 0);
02119     }
02120
02121     /* Read longitude and latitude... */
02122     NC_GET_DOUBLE("LON", atm->lon, 1);
02123     NC_GET_DOUBLE("LAT", atm->lat, 1);
02124
02125     /* Close file... */
02126     NC(nc_close(ncid));
02127
02128     /* Return success... */
02129     return 1;
02130 }
```

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

Read atmospheric data in netCDF format.

Definition at line 2134 of file [libtrac.c](#).

```
02137     {
02138
02139     int ncid, varid;
02140
02141     /* Open file... */
02142     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02143         return 0;
02144
02145     /* Get dimensions... */
02146     NC_INQ_DIM("obs", &atm->np, 1, NP);
02147
02148     /* Read geolocations... */
02149     NC_GET_DOUBLE("time", atm->time, 1);
```

```

02150 NC_GET_DOUBLE("press", atm->p, 1);
02151 NC_GET_DOUBLE("lon", atm->lon, 1);
02152 NC_GET_DOUBLE("lat", atm->lat, 1);
02153
02154 /* Read variables... */
02155 for (int iq = 0; iq < ctl->nq; iq++)
02156     NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
02157
02158 /* Close file... */
02159 NC(nc_close(ncid));
02160
02161 /* Return success... */
02162 return 1;
02163 }

```

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

Read climatological data.

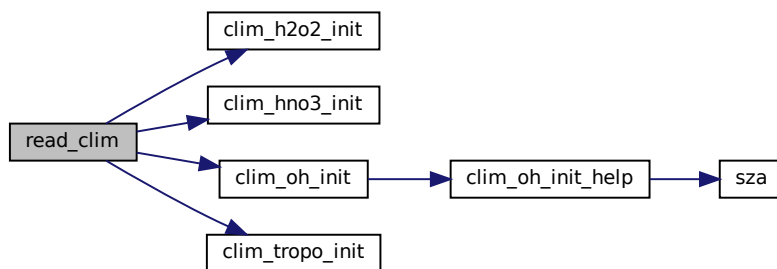
Definition at line 2167 of file libtrac.c.

```

02169 {
02170
02171     /* Set timer... */
02172     SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);
02173
02174     /* Init tropopause climatology... */
02175     clim_tropo_init(clim);
02176
02177     /* Init HNO3 climatology... */
02178     clim_hno3_init(clim);
02179
02180     /* Read OH climatology... */
02181     if (ctl->clim_oh_filename[0] != '-')
02182         clim_oh_init(ctl, clim);
02183
02184     /* Read H2O2 climatology... */
02185     if (ctl->clim_h2o2_filename[0] != '-')
02186         clim_h2o2_init(ctl, clim);
02187 }

```

Here is the call graph for this function:



```

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

```

Read control parameters.

Definition at line 2191 of file libtrac.c.

```

02195     {
02196
02197     /* Set timer... */
02198     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02199
02200     /* Write info... */
02201     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02202          "(executable: %s | version: %s | compiled: %s, %s)\n",
02203          argv[0], VERSION, __DATE__, __TIME__);
02204
02205     /* Initialize quantity indices... */
02206     ctl->qnt_idx = -1;
02207     ctl->qnt_ens = -1;
02208     ctl->qnt_stat = -1;
02209     ctl->qnt_m = -1;
02210     ctl->qnt_vmr = -1;
02211     ctl->qnt_rp = -1;
02212     ctl->qnt_rhop = -1;
02213     ctl->qnt_ps = -1;
02214     ctl->qnt_ts = -1;
02215     ctl->qnt_zs = -1;
02216     ctl->qnt_us = -1;
02217     ctl->qnt_vs = -1;
02218     ctl->qnt_pbl = -1;
02219     ctl->qnt_pt = -1;
02220     ctl->qnt_tt = -1;
02221     ctl->qnt_zt = -1;
02222     ctl->qnt_h2ot = -1;
02223     ctl->qnt_z = -1;
02224     ctl->qnt_p = -1;
02225     ctl->qnt_t = -1;
02226     ctl->qnt_rho = -1;
02227     ctl->qnt_u = -1;
02228     ctl->qnt_v = -1;
02229     ctl->qnt_w = -1;
02230     ctl->qnt_h2o = -1;
02231     ctl->qnt_o3 = -1;
02232     ctl->qnt_lwc = -1;
02233     ctl->qnt_iwc = -1;
02234     ctl->qnt_pct = -1;
02235     ctl->qnt_pcb = -1;
02236     ctl->qnt_cl = -1;
02237     ctl->qnt_plcl = -1;
02238     ctl->qnt_plfc = -1;
02239     ctl->qnt_pel = -1;
02240     ctl->qnt_cape = -1;
02241     ctl->qnt_cin = -1;
02242     ctl->qnt_hno3 = -1;
02243     ctl->qnt_oh = -1;
02244     ctl->qnt_vmrimpl = -1;
02245     ctl->qnt_mloss_oh = -1;
02246     ctl->qnt_mloss_h2o2 = -1;
02247     ctl->qnt_mloss_wet = -1;
02248     ctl->qnt_mloss_dry = -1;
02249     ctl->qnt_mloss_decay = -1;
02250     ctl->qnt_psat = -1;
02251     ctl->qnt_psice = -1;
02252     ctl->qnt_pw = -1;
02253     ctl->qnt_sh = -1;
02254     ctl->qnt_rh = -1;
02255     ctl->qnt_rhice = -1;
02256     ctl->qnt_theta = -1;
02257     ctl->qnt_zeta = -1;
02258     ctl->qnt_tvirt = -1;
02259     ctl->qnt_lapse = -1;
02260     ctl->qnt_vh = -1;
02261     ctl->qnt_vz = -1;
02262     ctl->qnt_pv = -1;
02263     ctl->qnt_tdew = -1;
02264     ctl->qnt_tice = -1;
02265     ctl->qnt_tsts = -1;
02266     ctl->qnt_tnat = -1;
02267
02268     /* Read quantities... */

```

```

02269     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02270     if (ctl->nq > NQ)
02271         ERRMSG("Too many quantities!");
02272     for (int iq = 0; iq < ctl->nq; iq++) {
02273
02274         /* Read quantity name and format... */
02275         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02276         scan_ctl(filename, argc, argv, "QNT_LONGNAME", iq, ctl->qnt_name[iq],
02277             ctl->qnt_longname[iq]);
02278         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02279             ctl->qnt_format[iq]);
02280
02281         /* Try to identify quantity... */
02282         SET_QNT(qnt_idx, "idx", "particle index", "-")
02283         SET_QNT(qnt_ens, "ens", "ensemble index", "-")
02284         SET_QNT(qnt_stat, "stat", "station flag", "-")
02285         SET_QNT(qnt_m, "m", "mass", "kg")
02286         SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
02287         SET_QNT(qnt_rp, "rp", "particle radius", "microns")
02288         SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
02289         SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
02290         SET_QNT(qnt_ts, "ts", "surface temperature", "K")
02291         SET_QNT(qnt_zs, "zs", "surface height", "km")
02292         SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
02293         SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
02294         SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
02295         SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
02296         SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
02297         SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
02298         SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
02299         SET_QNT(qnt_z, "z", "geopotential height", "km")
02300         SET_QNT(qnt_p, "p", "pressure", "hPa")
02301         SET_QNT(qnt_t, "t", "temperature", "K")
02302         SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
02303         SET_QNT(qnt_u, "u", "zonal wind", "m/s")
02304         SET_QNT(qnt_v, "v", "meridional wind", "m/s")
02305         SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
02306         SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
02307         SET_QNT(qnt_o3, "o3", "ozone", "ppv")
02308         SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
02309         SET_QNT(qnt_lwc, "lwc", "cloud liquid water content", "kg/kg")
02310         SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
02311         SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
02312         SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
02313         SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
02314         SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
02315         SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
02316         SET_QNT(qnt_cape, "cape", "convective available potential energy",
02317             "J/kg")
02318         SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
02319         SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
02320         SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
02321         SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
02322         SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
02323         SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
02324             "kg")
02325         SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
02326             "kg")
02327         SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
02328             "kg")
02329         SET_QNT(qnt_mloss_decay, "mloss_decay",
02330             "mass loss due to exponential decay", "kg")
02331         SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
02332         SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
02333         SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
02334         SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
02335         SET_QNT(qnt_rh, "rh", "relative humidity", "%")
02336         SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%")
02337         SET_QNT(qnt_theta, "theta", "potential temperature", "K")
02338         SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
02339         SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
02340         SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
02341         SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
02342         SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
02343         SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
02344         SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
02345         SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
02346         SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
02347         SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
02348         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02349     }
02350
02351     /* netCDF I/O parameters... */
02352     ctl->chunkszhint =
02353         (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02354             NULL);
02355     ctl->read_mode =

```

```

02356     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02357
02358 /* Vertical coordinates and velocities... */
02359 ctl->vert_coord_ap =
02360     (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
02361 ctl->vert_coord_met =
02362     (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
02363 ctl->vert_vel =
02364     (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
02365 ctl->clams_met_data =
02366     (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);
02367
02368 /* Time steps of simulation... */
02369 ctl->direction =
02370     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02371 if (ctl->direction != -1 && ctl->direction != 1)
02372     ERRMSG("Set DIRECTION to -1 or 1!");
02373 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02374 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02375
02376 /* Meteo data... */
02377 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02378 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02379 ctl->met_type =
02380     (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
02381 ctl->met_nc_scale =
02382     (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", NULL);
02383 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02384 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02385 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02386 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02387     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02388 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02389 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02390 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02391 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02392     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02393 ctl->met_detrend =
02394     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02395 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02396 if (ctl->met_np > EP)
02397     ERRMSG("Too many levels!");
02398 for (int ip = 0; ip < ctl->met_np; ip++)
02399     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02400 ctl->met_geopot_sx =
02401     (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02402 ctl->met_geopot_sy =
02403     (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02404 ctl->met_relhun =
02405     (int) scan_ctl(filename, argc, argv, "MET_RELHUM", -1, "0", NULL);
02406 ctl->met_tropo =
02407     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02408 if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02409     ERRMSG("Set MET_TROPO = 0 ... 5!");
02410 ctl->met_tropo_lapse =
02411     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02412 ctl->met_tropo_nlev =
02413     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02414 ctl->met_tropo_lapse_sep =
02415     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02416 ctl->met_tropo_nlev_sep =
02417     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02418     NULL);
02419 ctl->met_tropo_pv =
02420     scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02421 ctl->met_tropo_theta =
02422     scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02423 ctl->met_tropo_spline =
02424     (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02425 ctl->met_cloud =
02426     (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02427 if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02428     ERRMSG("Set MET_CLOUD = 0 ... 3!");
02429 ctl->met_cloud_min =
02430     scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
02431 ctl->met_dt_out =
02432     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02433 ctl->met_cache =
02434     (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02435
02436 /* Sorting... */
02437 ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);
02438
02439 /* Isosurface parameters... */
02440 ctl->isosurf =
02441     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02442 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

```

```

02443
02444 /* Advection parameters... */
02445 ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
02446 if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
02447     ERRMSG("Set ADVECT to 1, 2, or 4!");
02448 ctl->reflect =
02449     (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02450
02451 /* Diffusion parameters... */
02452 ctl->turb_dx_trop =
02453     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02454 ctl->turb_dx_strat =
02455     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02456 ctl->turb_dz_trop =
02457     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02458 ctl->turb_dz_strat =
02459     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02460 ctl->turb_mesox =
02461     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02462 ctl->turb_mesoz =
02463     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02464
02465 /* Convection... */
02466 ctl->conv_cape =
02467     scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02468 ctl->conv_cin =
02469     scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02470 ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02471 ctl->conv_mix =
02472     (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "1", NULL);
02473 ctl->conv_mix_bot =
02474     (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02475 ctl->conv_mix_top =
02476     (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02477
02478 /* Boundary conditions... */
02479 ctl->bound_mass =
02480     scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02481 ctl->bound_mass_trend =
02482     scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
02483 ctl->bound_vmr =
02484     scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02485 ctl->bound_vmr_trend =
02486     scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
02487 ctl->bound_lat0 =
02488     scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02489 ctl->bound_lat1 =
02490     scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02491 ctl->bound_p0 =
02492     scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02493 ctl->bound_p1 =
02494     scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02495 ctl->bound_dps =
02496     scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02497 ctl->bound_dzs =
02498     scan_ctl(filename, argc, argv, "BOUND_DZS", -1, "-999", NULL);
02499 ctl->bound_zetas =
02500     scan_ctl(filename, argc, argv, "BOUND_ZETAS", -1, "-999", NULL);
02501 ctl->bound_pbl =
02502     (int) scan_ctl(filename, argc, argv, "BOUND_PBL", -1, "0", NULL);
02503
02504 /* Species parameters... */
02505 scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02506 if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02507     ctl->molmass = 120.907;
02508     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
02509     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
02510 } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02511     ctl->molmass = 137.359;
02512     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
02513     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
02514 } else if (strcasecmp(ctl->species, "CH4") == 0) {
02515     ctl->molmass = 16.043;
02516     ctl->oh_chem_reaction = 2;
02517     ctl->oh_chem[0] = 2.45e-12;
02518     ctl->oh_chem[1] = 1775;
02519     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
02520     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02521 } else if (strcasecmp(ctl->species, "CO") == 0) {
02522     ctl->molmass = 28.01;
02523     ctl->oh_chem_reaction = 3;
02524     ctl->oh_chem[0] = 6.9e-33;
02525     ctl->oh_chem[1] = 2.1;
02526     ctl->oh_chem[2] = 1.1e-12;
02527     ctl->oh_chem[3] = -1.3;
02528     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
02529     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;

```

```
02530 } else if (strcasecmp(ctl->species, "CO2") == 0) {
02531     ctl->molmass = 44.009;
02532     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
02533     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02534 } else if (strcasecmp(ctl->species, "H2O") == 0) {
02535     ctl->molmass = 18.01528;
02536 } else if (strcasecmp(ctl->species, "N2O") == 0) {
02537     ctl->molmass = 44.013;
02538     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
02539     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
02540 } else if (strcasecmp(ctl->species, "NH3") == 0) {
02541     ctl->molmass = 17.031;
02542     ctl->oh_chem_reaction = 2;
02543     ctl->oh_chem[0] = 1.7e-12;
02544     ctl->oh_chem[1] = 710;
02545     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
02546     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
02547 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02548     ctl->molmass = 63.012;
02549     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
02550     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
02551 } else if (strcasecmp(ctl->species, "NO") == 0) {
02552     ctl->molmass = 30.006;
02553     ctl->oh_chem_reaction = 3;
02554     ctl->oh_chem[0] = 7.1e-31;
02555     ctl->oh_chem[1] = 2.6;
02556     ctl->oh_chem[2] = 3.6e-11;
02557     ctl->oh_chem[3] = 0.1;
02558     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
02559     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02560 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02561     ctl->molmass = 46.005;
02562     ctl->oh_chem_reaction = 3;
02563     ctl->oh_chem[0] = 1.8e-30;
02564     ctl->oh_chem[1] = 3.0;
02565     ctl->oh_chem[2] = 2.8e-11;
02566     ctl->oh_chem[3] = 0.0;
02567     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
02568     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02569 } else if (strcasecmp(ctl->species, "O3") == 0) {
02570     ctl->molmass = 47.997;
02571     ctl->oh_chem_reaction = 2;
02572     ctl->oh_chem[0] = 1.7e-12;
02573     ctl->oh_chem[1] = 940;
02574     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
02575     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
02576 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02577     ctl->molmass = 146.048;
02578     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
02579     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
02580 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02581     ctl->molmass = 64.066;
02582     ctl->oh_chem_reaction = 3;
02583     ctl->oh_chem[0] = 2.9e-31;
02584     ctl->oh_chem[1] = 4.1;
02585     ctl->oh_chem[2] = 1.7e-12;
02586     ctl->oh_chem[3] = -0.2;
02587     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
02588     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
02589 } else {
02590     ctl->molmass =
02591         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02592     ctl->oh_chem_reaction =
02593         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02594     ctl->h2o2_chem_reaction =
02595         (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
02596             NULL);
02597     for (int ip = 0; ip < 4; ip++)
02598         ctl->oh_chem[ip] =
02599             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02600     ctl->dry_depo_vdep =
02601         scan_ctl(filename, argc, argv, "DRY_DEPO_VDEP", -1, "0", NULL);
02602     ctl->dry_depo_dp =
02603         scan_ctl(filename, argc, argv, "DRY_DEPO_DP", -1, "30", NULL);
02604     ctl->wet_depo_ic_a =
02605         scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
02606     ctl->wet_depo_ic_b =
02607         scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
02608     ctl->wet_depo_bc_a =
02609         scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
02610     ctl->wet_depo_bc_b =
02611         scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
02612     for (int ip = 0; ip < 3; ip++)
02613         ctl->wet_depo_ic_h[ip] =
02614             scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
02615     for (int ip = 0; ip < 1; ip++)
02616         ctl->wet_depo_bc_h[ip] =
```

```

02617         scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
02618     }
02619
02620     /* Wet deposition... */
02621     ctl->wet_depo_pre[0] =
02622         scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
02623     ctl->wet_depo_pre[1] =
02624         scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
02625     ctl->wet_depo_ic_ret_ratio =
02626         scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
02627     ctl->wet_depo_bc_ret_ratio =
02628         scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);
02629
02630     /* OH chemistry... */
02631     ctl->oh_chem_beta =
02632         scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
02633     scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
02634         ".../data/clams_radical_species.nc", ctl->clim_oh_filename);
02635
02636     /* H2O2 chemistry... */
02637     ctl->h2o2_chem_cc =
02638         scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
02639     scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
02640         ".../data/cams_H2O2.nc", ctl->clim_h2o2_filename);
02641
02642     /* Chemistry grid... */
02643     ctl->chemgrid_z0 =
02644         scan_ctl(filename, argc, argv, "CHEMGRID_Z0", -1, "0", NULL);
02645     ctl->chemgrid_z1 =
02646         scan_ctl(filename, argc, argv, "CHEMGRID_Z1", -1, "100", NULL);
02647     ctl->chemgrid_nz =
02648         (int) scan_ctl(filename, argc, argv, "CHEMGRID_NZ", -1, "1", NULL);
02649     ctl->chemgrid_lon0 =
02650         scan_ctl(filename, argc, argv, "CHEMGRID_LON0", -1, "-180", NULL);
02651     ctl->chemgrid_lon1 =
02652         scan_ctl(filename, argc, argv, "CHEMGRID_LON1", -1, "180", NULL);
02653     ctl->chemgrid_nx =
02654         (int) scan_ctl(filename, argc, argv, "CHEMGRID_NX", -1, "360", NULL);
02655     ctl->chemgrid_lat0 =
02656         scan_ctl(filename, argc, argv, "CHEMGRID_LAT0", -1, "-90", NULL);
02657     ctl->chemgrid_lat1 =
02658         scan_ctl(filename, argc, argv, "CHEMGRID_LAT1", -1, "90", NULL);
02659     ctl->chemgrid_ny =
02660         (int) scan_ctl(filename, argc, argv, "CHEMGRID_NY", -1, "180", NULL);
02661
02662     /* Exponential decay... */
02663     ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02664     ctl->tdec_strat =
02665         scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02666
02667     /* PSC analysis... */
02668     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02669     ctl->psc_hno3 =
02670         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02671
02672     /* Output of atmospheric data... */
02673     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02674     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02675     ctl->atm_dt_out =
02676         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02677     ctl->atm_filter =
02678         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02679     ctl->atm_stride =
02680         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02681     ctl->atm_type =
02682         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02683
02684     /* Output of CSI data... */
02685     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02686     ctl->csi_dt_out =
02687         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02688     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02689     ctl->csi_obsmin =
02690         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02691     ctl->csi_modmin =
02692         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02693     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02694     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02695     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02696     ctl->csi_lon0 =
02697         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02698     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02699     ctl->csi_nx =
02700         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02701     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02702     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02703     ctl->csi_ny =

```



```

02704     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02705
02706     /* Output of ensemble data... */
02707     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02708     ctl->ens_dt_out =
02709         scan_ctl(filename, argc, argv, "ENS_DT_OUT", -1, "86400", NULL);
02710
02711     /* Output of grid data... */
02712     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02713         ctl->grid_basename);
02714     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02715     ctl->grid_dt_out =
02716         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02717     ctl->grid_sparse =
02718         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02719     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02720     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02721     ctl->grid_nz =
02722         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02723     ctl->grid_lon0 =
02724         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02725     ctl->grid_lon1 =
02726         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02727     ctl->grid_nx =
02728         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02729     ctl->grid_lat0 =
02730         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02731     ctl->grid_lat1 =
02732         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02733     ctl->grid_ny =
02734         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02735     ctl->grid_type =
02736         (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", NULL);
02737
02738     /* Output of profile data... */
02739     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02740         ctl->prof_basename);
02741     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02742     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02743     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02744     ctl->prof_nz =
02745         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02746     ctl->prof_lon0 =
02747         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02748     ctl->prof_lon1 =
02749         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02750     ctl->prof_nx =
02751         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02752     ctl->prof_lat0 =
02753         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02754     ctl->prof_lat1 =
02755         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02756     ctl->prof_ny =
02757         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02758
02759     /* Output of sample data... */
02760     scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02761         ctl->sample_basename);
02762     scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02763         ctl->sample_obsfile);
02764     ctl->sample_dx =
02765         scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02766     ctl->sample_dz =
02767         scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02768
02769     /* Output of station data... */
02770     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02771         ctl->stat_basename);
02772     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02773     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02774     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02775     ctl->stat_t0 =
02776         scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02777     ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02778 }

```

Here is the call graph for this function:



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

Read meteo data file.

Definition at line 2782 of file libtrac.c.

```

02786     {
02787
02788     /* Write info... */
02789     LOG(1, "Read meteo data: %s", filename);
02790
02791     /* Read netCDF data... */
02792     if (ctl->met_type == 0) {
02793
02794         int ncid;
02795
02796         /* Open netCDF file... */
02797         if (nc_open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02798             NC_NOERR) {
02799             WARN("Cannot open file!");
02800             return 0;
02801         }
02802
02803         /* Read coordinates of meteo data... */
02804         read_met_grid(filename, ncid, ctl, met);
02805
02806         /* Read meteo data on vertical levels... */
02807         read_met_levels(ncid, ctl, met);
02808
02809         /* Extrapolate data for lower boundary... */
02810         read_met_extrapolate(met);
02811
02812         /* Read surface data... */
02813         read_met_surface(ncid, met, ctl);
02814
02815         /* Create periodic boundary conditions... */
02816         read_met_periodic(met);
02817
02818         /* Downsampling... */
02819         read_met_sample(ctl, met);
02820
02821         /* Calculate geopotential heights... */
02822         read_met_geopot(ctl, met);
02823
02824         /* Calculate potential vorticity... */
02825         read_met_pv(met);
02826
02827         /* Calculate boundary layer data... */
02828         read_met_pbl(met);
02829
02830         /* Calculate tropopause data... */
02831         read_met_tropo(ctl, clim, met);
02832
02833         /* Calculate cloud properties... */
02834         read_met_cloud(ctl, met);
  
```

```

02835
02836     /* Calculate convective available potential energy... */
02837     read_met_cape(clim, met);
02838
02839     /* Detrending... */
02840     read_met_detrend(ctl, met);
02841
02842     /* Close file... */
02843     NC(nc_close(ncid));
02844 }
02845
02846 /* Read binary data... */
02847 else if (ctl->met_type >= 1 && ctl->met_type <= 4) {
02848
02849     FILE *in;
02850
02851     double r;
02852
02853     int year, mon, day, hour, min, sec;
02854
02855     /* Set timer... */
02856     SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);
02857
02858     /* Open file... */
02859     if (!(in = fopen(filename, "r"))) {
02860         WARN("Cannot open file!");
02861         return 0;
02862     }
02863
02864     /* Check type of binary data... */
02865     int met_type;
02866     FREAD(&met_type, int,
02867          1,
02868          in);
02869     if (met_type != ctl->met_type)
02870         ERRMSG("Wrong MET_TYPE of binary data!");
02871
02872     /* Check version of binary data... */
02873     int version;
02874     FREAD(&version, int,
02875          1,
02876          in);
02877     if (version != 100)
02878         ERRMSG("Wrong version of binary data!");
02879
02880     /* Read time... */
02881     FREAD(&met->time, double,
02882          1,
02883          in);
02884     jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
02885     LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
02886         met->time, year, mon, day, hour, min);
02887     if (year < 1900 || year > 2100 || mon < 1 || mon > 12
02888         || day < 1 || day > 31 || hour < 0 || hour > 23)
02889         ERRMSG("Error while reading time!");
02890
02891     /* Read dimensions... */
02892     FREAD(&met->nx, int,
02893          1,
02894          in);
02895     LOG(2, "Number of longitudes: %d", met->nx);
02896     if (met->nx < 2 || met->nx > EX)
02897         ERRMSG("Number of longitudes out of range!");
02898
02899     FREAD(&met->ny, int,
02900          1,
02901          in);
02902     LOG(2, "Number of latitudes: %d", met->ny);
02903     if (met->ny < 2 || met->ny > EY)
02904         ERRMSG("Number of latitudes out of range!");
02905
02906     FREAD(&met->np, int,
02907          1,
02908          in);
02909     LOG(2, "Number of levels: %d", met->np);
02910     if (met->np < 2 || met->np > EP)
02911         ERRMSG("Number of levels out of range!");
02912
02913     /* Read grid... */
02914     FREAD(met->lon, double,
02915          (size_t) met->nx,
02916          in);
02917     LOG(2, "Longitudes: %g, %g ... %g deg",
02918         met->lon[0], met->lon[1], met->lon[met->nx - 1]);
02919
02920     FREAD(met->lat, double,
02921          (size_t) met->ny,

```

```

02922         in);
02923     LOG(2, "Latitudes: %g, %g ... %g deg",
02924         met->lat[0], met->lat[1], met->lat[met->ny - 1]);
02925
02926     FREAD(met->p, double,
02927         (size_t) met->np,
02928         in);
02929     LOG(2, "Altitude levels: %g, %g ... %g km",
02930         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
02931     LOG(2, "Pressure levels: %g, %g ... %g hPa",
02932         met->p[0], met->p[1], met->p[met->np - 1]);
02933
02934     /* Read surface data... */
02935     read_met_bin_2d(in, met, met->ps, "PS");
02936     read_met_bin_2d(in, met, met->ts, "TS");
02937     read_met_bin_2d(in, met, met->zs, "ZS");
02938     read_met_bin_2d(in, met, met->us, "US");
02939     read_met_bin_2d(in, met, met->vs, "VS");
02940     read_met_bin_2d(in, met, met->pbl, "PBL");
02941     read_met_bin_2d(in, met, met->pt, "PT");
02942     read_met_bin_2d(in, met, met->tt, "TT");
02943     read_met_bin_2d(in, met, met->zt, "ZT");
02944     read_met_bin_2d(in, met, met->h2ot, "H2OT");
02945     read_met_bin_2d(in, met, met->pct, "PCT");
02946     read_met_bin_2d(in, met, met->pcb, "PCB");
02947     read_met_bin_2d(in, met, met->cl, "CL");
02948     read_met_bin_2d(in, met, met->plcl, "PLCL");
02949     read_met_bin_2d(in, met, met->plfc, "PLFC");
02950     read_met_bin_2d(in, met, met->pel, "PEL");
02951     read_met_bin_2d(in, met, met->cape, "CAPE");
02952     read_met_bin_2d(in, met, met->cin, "CIN");
02953
02954     /* Read level data... */
02955     read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
02956     read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
02957     read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
02958     read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
02959     read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
02960     read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
02961     read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
02962     read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
02963     read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
02964     read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);
02965
02966     /* Read final flag... */
02967     int final;
02968     FREAD(&final, int,
02969         1,
02970         in);
02971     if (final != 999)
02972         ERRMSG("Error while reading binary data!");
02973
02974     /* Close file... */
02975     fclose(in);
02976 }
02977
02978 /* Not implemented... */
02979 else
02980     ERRMSG("MET_TYPE not implemented!");
02981
02982 /* Copy wind data to cache... */
02983 #ifdef UVW
02984 #pragma omp parallel for default(shared) collapse(2)
02985     for (int ix = 0; ix < met->nx; ix++)
02986         for (int iy = 0; iy < met->ny; iy++)
02987             for (int ip = 0; ip < met->np; ip++) {
02988                 met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
02989                 met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
02990                 met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
02991             }
02992 #endif
02993
02994 /* Return success... */
02995 return 1;
02996 }

```



```

03011
03012 /* Read uncompressed... */
03013 LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
03014 FREAD(help, float,
03015       (size_t) (met->nx * met->ny),
03016       in);
03017
03018 /* Copy data... */
03019 for (int ix = 0; ix < met->nx; ix++)
03020     for (int iy = 0; iy < met->ny; iy++)
03021         var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];
03022
03023 /* Free... */
03024 free(help);
03025 }

```

5.21.2.40 read_met_bin_3d() void read_met_bin_3d (

```

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

```

Read 3-D meteo variable.

Definition at line 3029 of file libtrac.c.

```

03036 {
03037
03038     float *help;
03039
03040     /* Allocate... */
03041     ALLOC(help, float,
03042           EX * EY * EP);
03043
03044     /* Read uncompressed data... */
03045     if (ctl->met_type == 1) {
03046         LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
03047         FREAD(help, float,
03048               (size_t) (met->nx * met->ny * met->np),
03049               in);
03050     }
03051
03052     /* Read packed data... */
03053     else if (ctl->met_type == 2)
03054         compress_pack(varname, help, (size_t) (met->ny * met->nx),
03055                       (size_t) met->np, 1, in);
03056
03057     /* Read zfp data... */
03058     else if (ctl->met_type == 3) {
03059 #ifdef ZFP
03060         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
03061                     tolerance, 1, in);
03062 #else
03063         ERRMSG("zfp compression not supported!");
03064         LOG(3, "%d %g", precision, tolerance);
03065 #endif
03066     }
03067
03068     /* Read zstd data... */
03069     else if (ctl->met_type == 4) {
03070 #ifdef ZSTD
03071         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
03072                     in);
03073 #else
03074         ERRMSG("zstd compression not supported!");
03075 #endif
03076     }
03077
03078     /* Copy data... */
03079 #pragma omp parallel for default(shared) collapse(2)
03080     for (int ix = 0; ix < met->nx; ix++)
03081         for (int iy = 0; iy < met->ny; iy++)
03082             for (int ip = 0; ip < met->np; ip++)
03083                 var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];

```

```

03084
03085     /* Free... */
03086     free(help);
03087 }

```

Here is the call graph for this function:



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

Calculate convective available potential energy.

Definition at line 3091 of file libtrac.c.

```

03093     {
03094
03095     /* Set timer... */
03096     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
03097     LOG(2, "Calculate CAPE...");
03098
03099     /* Vertical spacing (about 100 m)... */
03100     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
03101
03102     /* Loop over columns... */
03103     #pragma omp parallel for default(shared) collapse(2)
03104     for (int ix = 0; ix < met->nx; ix++)
03105         for (int iy = 0; iy < met->ny; iy++) {
03106
03107             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
03108             int n = 0;
03109             double h2o = 0, t, theta = 0;
03110             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
03111             double ptop = pbot - 50.;
03112             for (int ip = 0; ip < met->np; ip++) {
03113                 if (met->p[ip] <= pbot) {
03114                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
03115                     h2o += met->h2o[ix][iy][ip];
03116                     n++;
03117                 }
03118                 if (met->p[ip] < ptop && n > 0)
03119                     break;
03120             }
03121             theta /= n;
03122             h2o /= n;
03123
03124             /* Cannot compute anything if water vapor is missing... */
03125             met->plcl[ix][iy] = GSL_NAN;
03126             met->plfc[ix][iy] = GSL_NAN;
03127             met->pel[ix][iy] = GSL_NAN;
03128             met->cape[ix][iy] = GSL_NAN;
03129             met->cin[ix][iy] = GSL_NAN;
03130             if (h2o <= 0)
03131                 continue;
03132
03133             /* Find lifted condensation level (LCL)... */
03134             ptop = P(20.);
03135             pbot = met->ps[ix][iy];
03136             do {
03137                 met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
03138                 t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
03139                 if (RH(met->plcl[ix][iy], t, h2o) > 100.)

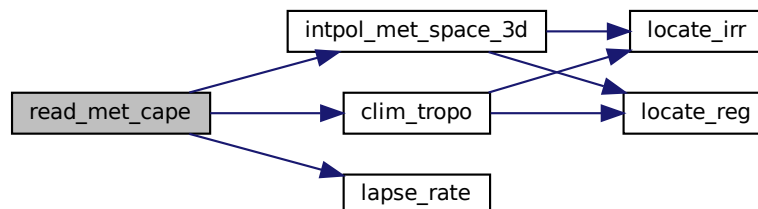
```

```

03140     ptop = met->plcl[ix][iy];
03141     else
03142     pbot = met->plcl[ix][iy];
03143 } while (pbot - ptop > 0.1);
03144
03145 /* Calculate CIN up to LCL... */
03146 INTPOL_INIT;
03147 double dcaper, dz, h2o_env, t_env;
03148 double p = met->ps[ix][iy];
03149 met->cape[ix][iy] = met->cin[ix][iy] = 0;
03150 do {
03151     dz = dz0 * TVIRT(t, h2o);
03152     p /= pfac;
03153     t = theta / pow(1000. / p, 0.286);
03154     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03155         &t_env, ci, cw, 1);
03156     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03157         &h2o_env, ci, cw, 0);
03158     dcaper = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03159         TVIRT(t_env, h2o_env) * dz;
03160     if (dcaper < 0)
03161         met->cin[ix][iy] += fabsf((float) dcaper);
03162 } while (p > met->plcl[ix][iy]);
03163
03164 /* Calculate level of free convection (LFC), equilibrium level (EL),
03165    and convective available potential energy (CAPE)... */
03166 dcaper = 0;
03167 p = met->plcl[ix][iy];
03168 t = theta / pow(1000. / p, 0.286);
03169 ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
03170 do {
03171     dz = dz0 * TVIRT(t, h2o);
03172     p /= pfac;
03173     t -= lapse_rate(t, h2o) * dz;
03174     double psat = PSAT(t);
03175     h2o = psat / (p - (1. - EPS) * psat);
03176     intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03177         &t_env, ci, cw, 1);
03178     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03179         &h2o_env, ci, cw, 0);
03180     double dcaper_old = dcaper;
03181     dcaper = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03182         TVIRT(t_env, h2o_env) * dz;
03183     if (dcaper > 0) {
03184         met->cape[ix][iy] += (float) dcaper;
03185         if (!isfinite(met->plfc[ix][iy]))
03186             met->plfc[ix][iy] = (float) p;
03187     } else if (dcaper_old > 0)
03188         met->pel[ix][iy] = (float) p;
03189     if (dcaper < 0 && !isfinite(met->plfc[ix][iy]))
03190         met->cin[ix][iy] += fabsf((float) dcaper);
03191 } while (p > ptop);
03192
03193 /* Check results... */
03194 if (!isfinite(met->plfc[ix][iy]))
03195     met->cin[ix][iy] = GSL_NAN;
03196 }
03197 }

```

Here is the call graph for this function:



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

Calculate cloud properties.

Definition at line 3201 of file libtrac.c.

```
03203     {
03204
03205     /* Set timer... */
03206     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
03207     LOG(2, "Calculate cloud data...");
03208
03209     /* Loop over columns... */
03210     #pragma omp parallel for default(shared) collapse(2)
03211     for (int ix = 0; ix < met->nx; ix++)
03212         for (int iy = 0; iy < met->ny; iy++) {
03213
03214         /* Init... */
03215         met->pct[ix][iy] = GSL_NAN;
03216         met->pcb[ix][iy] = GSL_NAN;
03217         met->cl[ix][iy] = 0;
03218
03219         /* Loop over pressure levels... */
03220         for (int ip = 0; ip < met->np - 1; ip++) {
03221
03222             /* Check pressure... */
03223             if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
03224                 continue;
03225
03226             /* Check ice water and liquid water content... */
03227             if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
03228                 || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {
03229
03230                 /* Get cloud top pressure ... */
03231                 met->pct[ix][iy]
03232                     = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
03233
03234                 /* Get cloud bottom pressure ... */
03235                 if (!isfinite(met->pcb[ix][iy]))
03236                     met->pcb[ix][iy]
03237                         = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
03238             }
03239
03240             /* Get cloud water... */
03241             met->cl[ix][iy] += (float)
03242                 (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
03243                     + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
03244                 * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
03245         }
03246     }
03247 }
```

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

Apply detrending method to temperature and winds.

Definition at line 3251 of file libtrac.c.

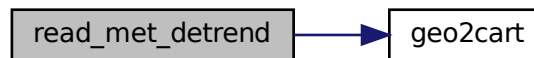
```
03253     {
03254
03255     met_t *help;
03256
03257     /* Check parameters... */
03258     if (ctl->met_detrend <= 0)
03259         return;
03260
03261     /* Set timer... */
03262     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
03263     LOG(2, "Detrend meteo data...");
03264
03265     /* Allocate... */
03266     ALLOC(help, met_t, 1);
03267
03268     /* Calculate standard deviation... */
```

```

03269 double sigma = ctl->met_detrend / 2.355;
03270 double tssq = 2. * SQR(sigma);
03271
03272 /* Calculate box size in latitude... */
03273 int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03274 sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03275
03276 /* Calculate background... */
03277 #pragma omp parallel for default(shared) collapse(2)
03278 for (int ix = 0; ix < met->nx; ix++) {
03279     for (int iy = 0; iy < met->ny; iy++) {
03280
03281         /* Calculate Cartesian coordinates... */
03282         double x0[3];
03283         geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03284
03285         /* Calculate box size in longitude... */
03286         int sx =
03287             (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03288                 fabs(met->lon[1] - met->lon[0]));
03289         sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03290
03291         /* Init... */
03292         float wsum = 0;
03293         for (int ip = 0; ip < met->np; ip++) {
03294             help->t[ix][iy][ip] = 0;
03295             help->u[ix][iy][ip] = 0;
03296             help->v[ix][iy][ip] = 0;
03297             help->w[ix][iy][ip] = 0;
03298         }
03299
03300         /* Loop over neighboring grid points... */
03301         for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03302             int ix3 = ix2;
03303             if (ix3 < 0)
03304                 ix3 += met->nx;
03305             else if (ix3 >= met->nx)
03306                 ix3 -= met->nx;
03307             for (int iy2 = GSL_MAX(iy - sy, 0);
03308                 iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03309
03310                 /* Calculate Cartesian coordinates... */
03311                 double x1[3];
03312                 geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03313
03314                 /* Calculate weighting factor... */
03315                 float w = (float) exp(-DIST2(x0, x1) / tssq);
03316
03317                 /* Add data... */
03318                 wsum += w;
03319                 for (int ip = 0; ip < met->np; ip++) {
03320                     help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03321                     help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03322                     help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03323                     help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03324                 }
03325             }
03326         }
03327
03328         /* Normalize... */
03329         for (int ip = 0; ip < met->np; ip++) {
03330             help->t[ix][iy][ip] /= wsum;
03331             help->u[ix][iy][ip] /= wsum;
03332             help->v[ix][iy][ip] /= wsum;
03333             help->w[ix][iy][ip] /= wsum;
03334         }
03335     }
03336 }
03337
03338 /* Subtract background... */
03339 #pragma omp parallel for default(shared) collapse(3)
03340 for (int ix = 0; ix < met->nx; ix++)
03341     for (int iy = 0; iy < met->ny; iy++)
03342         for (int ip = 0; ip < met->np; ip++) {
03343             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03344             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03345             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03346             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03347         }
03348
03349 /* Free... */
03350 free(help);
03351 }

```

Here is the call graph for this function:



5.21.2.44 read_met_extrapolate() void read_met_extrapolate (
 met_t * met)

Extrapolate meteo data at lower boundary.

Definition at line 3355 of file libtrac.c.

```

03356     {
03357
03358     /* Set timer... */
03359     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03360     LOG(2, "Extrapolate meteo data...");
03361
03362     /* Loop over columns... */
03363     #pragma omp parallel for default(shared) collapse(2)
03364     for (int ix = 0; ix < met->nx; ix++)
03365         for (int iy = 0; iy < met->ny; iy++) {
03366
03367         /* Find lowest valid data point... */
03368         int ip0;
03369         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03370             if (!isfinite(met->t[ix][iy][ip0])
03371                 || !isfinite(met->u[ix][iy][ip0])
03372                 || !isfinite(met->v[ix][iy][ip0])
03373                 || !isfinite(met->w[ix][iy][ip0]))
03374                 break;
03375
03376         /* Extrapolate... */
03377         for (int ip = ip0; ip >= 0; ip--) {
03378             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03379             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03380             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03381             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03382             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03383             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03384             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03385             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03386         }
03387     }
03388 }
```

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

Calculate geopotential heights.

Definition at line 3392 of file libtrac.c.

```

03394     {
03395
03396     static float help[EP][EX][EY];
03397
```

```

03398 double logp[EP];
03399
03400 int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03401
03402 /* Set timer... */
03403 SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03404 LOG(2, "Calculate geopotential heights...");
03405
03406 /* Calculate log pressure... */
03407 #pragma omp parallel for default(shared)
03408 for (int ip = 0; ip < met->np; ip++)
03409     logp[ip] = log(met->p[ip]);
03410
03411 /* Apply hydrostatic equation to calculate geopotential heights... */
03412 #pragma omp parallel for default(shared) collapse(2)
03413 for (int ix = 0; ix < met->nx; ix++)
03414     for (int iy = 0; iy < met->ny; iy++) {
03415
03416         /* Get surface height and pressure... */
03417         double zs = met->zs[ix][iy];
03418         double lnps = log(met->ps[ix][iy]);
03419
03420         /* Get temperature and water vapor vmr at the surface... */
03421         int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03422         double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03423             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03424         double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03425             met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03426
03427         /* Upper part of profile... */
03428         met->z[ix][iy][ip0 + 1]
03429             = (float) (zs +
03430                 ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03431                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03432         for (int ip = ip0 + 2; ip < met->np; ip++)
03433             met->z[ix][iy][ip]
03434                 = (float) (met->z[ix][iy][ip - 1] +
03435                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03436                         met->h2o[ix][iy][ip - 1], logp[ip],
03437                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03438
03439         /* Lower part of profile... */
03440         met->z[ix][iy][ip0]
03441             = (float) (zs +
03442                 ZDIFF(lnps, ts, h2os, logp[ip0],
03443                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03444         for (int ip = ip0 - 1; ip >= 0; ip--)
03445             met->z[ix][iy][ip]
03446                 = (float) (met->z[ix][iy][ip + 1] +
03447                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03448                         met->h2o[ix][iy][ip + 1], logp[ip],
03449                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03450     }
03451
03452 /* Check control parameters... */
03453 if (dx == 0 || dy == 0)
03454     return;
03455
03456 /* Default smoothing parameters... */
03457 if (dx < 0 || dy < 0) {
03458     if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03459         dx = 3;
03460         dy = 2;
03461     } else {
03462         dx = 6;
03463         dy = 4;
03464     }
03465 }
03466
03467 /* Calculate weights for smoothing... */
03468 float ws[dx + 1][dy + 1];
03469 #pragma omp parallel for default(shared) collapse(2)
03470 for (int ix = 0; ix <= dx; ix++)
03471     for (int iy = 0; iy < dy; iy++)
03472         ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03473             * (1.0f - (float) iy / (float) dy);
03474
03475 /* Copy data... */
03476 #pragma omp parallel for default(shared) collapse(3)
03477 for (int ix = 0; ix < met->nx; ix++)
03478     for (int iy = 0; iy < met->ny; iy++)
03479         for (int ip = 0; ip < met->np; ip++)
03480             help[ip][ix][iy] = met->z[ix][iy][ip];
03481
03482 /* Horizontal smoothing... */
03483 #pragma omp parallel for default(shared) collapse(3)
03484 for (int ip = 0; ip < met->np; ip++)

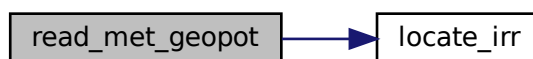
```

```

03485     for (int ix = 0; ix < met->nx; ix++)
03486     for (int iy = 0; iy < met->ny; iy++) {
03487         float res = 0, wsum = 0;
03488         int iy0 = GSL_MAX(iy - dy + 1, 0);
03489         int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03490         for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03491             int ix3 = ix2;
03492             if (ix3 < 0)
03493                 ix3 += met->nx;
03494             else if (ix3 >= met->nx)
03495                 ix3 -= met->nx;
03496             for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03497                 if (isfinite(help[ip][ix3][iy2])) {
03498                     float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03499                     res += w * help[ip][ix3][iy2];
03500                     wsum += w;
03501                 }
03502         }
03503         if (wsum > 0)
03504             met->z[ix][iy][ip] = res / wsum;
03505         else
03506             met->z[ix][iy][ip] = GSL_NAN;
03507     }
03508 }

```

Here is the call graph for this function:



5.21.2.46 read_met_grid() void read_met_grid (

```

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

```

Read coordinates of meteo data.

Definition at line 3512 of file libtrac.c.

```

03516     {
03517
03518         char levname[LEN], tstr[10];
03519
03520         double rtime, r2;
03521
03522         int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;
03523
03524         size_t np;
03525
03526         /* Set timer... */
03527         SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03528         LOG(2, "Read meteo grid information...");
03529
03530         /* MPTRAC meteo files... */
03531         if (ctl->clams_met_data == 0) {
03532
03533             /* Get time from filename... */
03534             size_t len = strlen(filename);
03535             sprintf(tstr, "%.4s", &filename[len - 16]);
03536             year = atoi(tstr);
03537             sprintf(tstr, "%.2s", &filename[len - 11]);
03538             mon = atoi(tstr);
03539             sprintf(tstr, "%.2s", &filename[len - 8]);

```

```

03540     day = atoi(tstr);
03541     sprintf(tstr, "%.2s", &filename[len - 5]);
03542     hour = atoi(tstr);
03543     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03544
03545     /* Check time information from data file... */
03546     if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03547         NC(nc_get_var_double(ncid, varid, &rttime));
03548         if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rttime) > 1.0)
03549             WARN("Time information in meteo file does not match filename!");
03550     } else
03551         WARN("Time information in meteo file is missing!");
03552 }
03553
03554 /* CLaMS meteo files... */
03555 else {
03556
03557     /* Read time from file... */
03558     NC_GET_DOUBLE("time", &rttime, 0);
03559
03560     /* Get time from filename (considering the century)... */
03561     if (rttime < 0)
03562         sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
03563     else
03564         sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
03565     year = atoi(tstr);
03566     sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
03567     mon = atoi(tstr);
03568     sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
03569     day = atoi(tstr);
03570     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03571     hour = atoi(tstr);
03572     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03573 }
03574
03575 /* Check time... */
03576 if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03577     || day < 1 || day > 31 || hour < 0 || hour > 23)
03578     ERRMSG("Cannot read time from filename!");
03579 jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03580 LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03581     met->time, year2, mon2, day2, hour2, min2);
03582
03583 /* Get grid dimensions... */
03584 NC_INQ_DIM("lon", &met->nx, 2, EX);
03585 LOG(2, "Number of longitudes: %d", met->nx);
03586
03587 NC_INQ_DIM("lat", &met->ny, 2, EY);
03588 LOG(2, "Number of latitudes: %d", met->ny);
03589
03590 if (ctl->vert_coord_meteo == 0) {
03591     int dimid;
03592     sprintf(levname, "lev");
03593     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR)
03594         sprintf(levname, "plev");
03595 } else
03596     sprintf(levname, "hybrid");
03597 NC_INQ_DIM(levname, &met->np, 1, EP);
03598 if (met->np == 1) {
03599     int dimid;
03600     sprintf(levname, "lev_2");
03601     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03602         sprintf(levname, "plev");
03603         nc_inq_dimid(ncid, levname, &dimid);
03604     }
03605     NC(nc_inq_dimlen(ncid, dimid, &np));
03606     met->np = (int) np;
03607 }
03608 LOG(2, "Number of levels: %d", met->np);
03609 if (met->np < 2 || met->np > EP)
03610     ERRMSG("Number of levels out of range!");
03611
03612 /* Read longitudes and latitudes... */
03613 NC_GET_DOUBLE("lon", met->lon, 1);
03614 LOG(2, "Longitudes: %g, %g ... %g deg",
03615     met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03616 NC_GET_DOUBLE("lat", met->lat, 1);
03617 LOG(2, "Latitudes: %g, %g ... %g deg",
03618     met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03619
03620 /* Read pressure levels... */
03621 if (ctl->met_np <= 0) {
03622     NC_GET_DOUBLE(levname, met->p, 1);
03623     for (int ip = 0; ip < met->np; ip++)
03624         met->p[ip] /= 100.;
03625     LOG(2, "Altitude levels: %g, %g ... %g km",
03626         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));

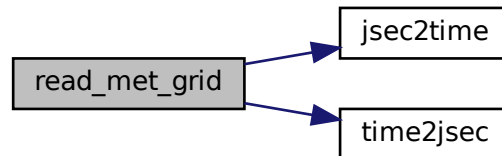
```

```

03627     LOG(2, "Pressure levels: %g, %g ... %g hPa",
03628         met->p[0], met->p[1], met->p[met->np - 1]);
03629 }
03630 }

```

Here is the call graph for this function:



5.21.2.47 read_met_levels() void read_met_levels (

 int ncid,

 ctl_t * ctl,

 met_t * met)

Read meteo data on vertical levels.

Definition at line 3634 of file libtrac.c.

```

03637     {
03638
03639     /* Set timer... */
03640     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03641     LOG(2, "Read level data...");
03642
03643     /* MPTRAC meteo data... */
03644     if (ctl->clams_met_data == 0) {
03645
03646     /* Read meteo data... */
03647     if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
03648         ERRMSG("Cannot read temperature!");
03649     if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03650         ERRMSG("Cannot read zonal wind!");
03651     if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03652         ERRMSG("Cannot read meridional wind!");
03653     if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
03654         WARN("Cannot read vertical velocity!");
03655     if (!read_met_nc_3d
03656         (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03657         WARN("Cannot read specific humidity!");
03658     if (!read_met_nc_3d
03659         (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03660         WARN("Cannot read ozone data!");
03661     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03662         if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03663             WARN("Cannot read cloud liquid water content!");
03664         if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03665             WARN("Cannot read cloud ice water content!");
03666     }
03667     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03668         if (!read_met_nc_3d
03669             (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03670              ctl->met_cloud == 2))
03671             WARN("Cannot read cloud rain water content!");
03672         if (!read_met_nc_3d
03673             (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03674              ctl->met_cloud == 2))
03675             WARN("Cannot read cloud snow water content!");
03676     }

```

```

03677     if (ctl->met_relhum) {
03678         if (!read_met_nc_3d(ncid, "rh", "RH", ctl, met, met->h2o, 0.01f, 1))
03679             WARN("Cannot read relative humidity!");
03680 #pragma omp parallel for default(shared) collapse(2)
03681         for (int ix = 0; ix < met->nx; ix++)
03682             for (int iy = 0; iy < met->ny; iy++)
03683                 for (int ip = 0; ip < met->np; ip++) {
03684                     double pw = met->h2o[ix][iy][ip] * PSAT(met->t[ix][iy][ip]);
03685                     met->h2o[ix][iy][ip] =
03686                         (float) (pw / (met->p[ip] - (1.0 - EPS) * pw));
03687                 }
03688     }
03689
03690     /* Transfer from model levels to pressure levels... */
03691     if (ctl->met_np > 0) {
03692
03693         /* Read pressure on model levels... */
03694         if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->p1, 0.01f, 1))
03695             ERRMSG("Cannot read pressure on model levels!");
03696
03697         /* Vertical interpolation from model to pressure levels... */
03698         read_met_ml2pl(ctl, met, met->t);
03699         read_met_ml2pl(ctl, met, met->u);
03700         read_met_ml2pl(ctl, met, met->v);
03701         read_met_ml2pl(ctl, met, met->w);
03702         read_met_ml2pl(ctl, met, met->h2o);
03703         read_met_ml2pl(ctl, met, met->o3);
03704         read_met_ml2pl(ctl, met, met->lwc);
03705         read_met_ml2pl(ctl, met, met->iwc);
03706
03707         /* Set new pressure levels... */
03708         met->np = ctl->met_np;
03709         for (int ip = 0; ip < met->np; ip++)
03710             met->p[ip] = ctl->met_p[ip];
03711     }
03712 }
03713
03714 /* CLaMS meteo data... */
03715 else if (ctl->clams_meteo_data == 1) {
03716
03717     /* Read meteorological data... */
03718     if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
03719         ERRMSG("Cannot read temperature!");
03720     if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03721         ERRMSG("Cannot read zonal wind!");
03722     if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03723         ERRMSG("Cannot read meridional wind!");
03724     if (!read_met_nc_3d(ncid, "w", "OMEGA", ctl, met, met->w, 0.01f, 1))
03725         WARN("Cannot read vertical velocity!");
03726     if (!read_met_nc_3d(ncid, "zeta", "ZETA", ctl, met, met->zeta, 1.0, 1))
03727         WARN("Cannot read ZETA in meteo data!");
03728     if (ctl->vert_vel == 1) {
03729         if (!read_met_nc_3d
03730             (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
03731              0.00001157407f, 1)) {
03732             if (!read_met_nc_3d
03733                 (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
03734                  0.00001157407f, 1)) {
03735                 WARN("Cannot read vertical velocity!");
03736             }
03737         }
03738     }
03739
03740     if (!read_met_nc_3d
03741         (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03742         WARN("Cannot read specific humidity!");
03743     if (!read_met_nc_3d
03744         (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03745         WARN("Cannot read ozone data!");
03746     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03747         if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03748             WARN("Cannot read cloud liquid water content!");
03749         if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03750             WARN("Cannot read cloud ice water content!");
03751     }
03752     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03753         if (!read_met_nc_3d
03754             (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03755              ctl->met_cloud == 2))
03756             WARN("Cannot read cloud rain water content!");
03757         if (!read_met_nc_3d
03758             (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03759              ctl->met_cloud == 2))
03760             WARN("Cannot read cloud snow water content!");
03761     }
03762
03763     /* Transfer from model levels to pressure levels... */

```

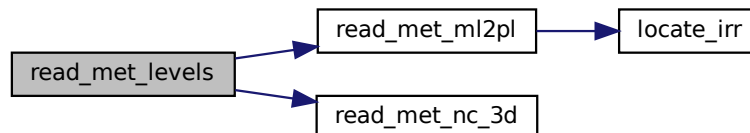


```

03764     if (ctl->met_np > 0) {
03765
03766         /* Read pressure on model levels... */
03767         if (!read_met_nc_3d(ncid, "p1", "PRESS", ctl, met, met->p1, 1.0, 1))
03768             ERRMSG("Cannot read pressure on model levels!");
03769
03770         /* Vertical interpolation from model to pressure levels... */
03771         read_met_ml2pl(ctl, met, met->t);
03772         read_met_ml2pl(ctl, met, met->u);
03773         read_met_ml2pl(ctl, met, met->v);
03774         read_met_ml2pl(ctl, met, met->w);
03775         read_met_ml2pl(ctl, met, met->h2o);
03776         read_met_ml2pl(ctl, met, met->o3);
03777         read_met_ml2pl(ctl, met, met->lwc);
03778         read_met_ml2pl(ctl, met, met->iwc);
03779         if (ctl->vert_vel == 1) {
03780             read_met_ml2pl(ctl, met, met->zeta);
03781             read_met_ml2pl(ctl, met, met->zeta_dot);
03782         }
03783
03784         /* Set new pressure levels... */
03785         met->np = ctl->met_np;
03786         for (int ip = 0; ip < met->np; ip++)
03787             met->p[ip] = ctl->met_p[ip];
03788
03789         /* Create a pressure field... */
03790         for (int i = 0; i < met->nx; i++)
03791             for (int j = 0; j < met->ny; j++)
03792                 for (int k = 0; k < met->np; k++) {
03793                     met->patp[i][j][k] = (float) met->p[k];
03794                 }
03795     }
03796 } else
03797     ERRMSG("Meteo data format unknown!");
03798
03799 /* Check ordering of pressure levels... */
03800 for (int ip = 1; ip < met->np; ip++)
03801     if (met->p[ip - 1] < met->p[ip])
03802         ERRMSG("Pressure levels must be descending!");
03803 }

```

Here is the call graph for this function:



5.21.2.48 read_met_ml2pl() void read_met_ml2pl (

```

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

```

Convert meteo data from model levels to pressure levels.

Definition at line 3807 of file libtrac.c.

```

03810     {
03811
03812         double aux[EP], p[EP];
03813
03814         /* Set timer... */
03815         SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03816         LOG(2, "Interpolate meteo data to pressure levels...");

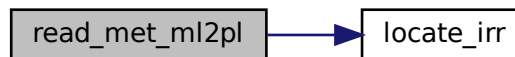
```

```

03817
03818  /* Loop over columns... */
03819 #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03820 for (int ix = 0; ix < met->nx; ix++)
03821   for (int iy = 0; iy < met->ny; iy++) {
03822
03823     /* Copy pressure profile... */
03824     for (int ip = 0; ip < met->np; ip++)
03825       p[ip] = met->pl[ix][iy][ip];
03826
03827     /* Interpolate... */
03828     for (int ip = 0; ip < ctl->met_np; ip++) {
03829       double pt = ctl->met_p[ip];
03830       if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03831         pt = p[0];
03832       else if ((pt > p[met->np - 1] && p[1] > p[0])
03833               || (pt < p[met->np - 1] && p[1] < p[0]))
03834         pt = p[met->np - 1];
03835       int ip2 = locate_irr(p, met->np, pt);
03836       aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03837                   p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03838     }
03839
03840     /* Copy data... */
03841     for (int ip = 0; ip < ctl->met_np; ip++)
03842       var[ix][iy][ip] = (float) aux[ip];
03843   }
03844 }

```

Here is the call graph for this function:



5.21.2.49 read_met_nc_2d() int read_met_nc_2d (

```

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

```

Read and convert 2D variable from meteo data file.

Definition at line 3848 of file libtrac.c.

```

03856     {
03857
03858     char varsel[LEN];
03859
03860     float offset, scalfac;
03861
03862     int varid;
03863
03864     /* Check if variable exists... */
03865     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03866       if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03867         WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
03868         return 0;
03869       } else {

```

```

03870     sprintf(varsel, "%s", varname2);
03871 } else
03872     sprintf(varsel, "%s", varname);
03873
03874 /* Read packed data... */
03875 if (ctl->met_nc_scale
03876     && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03877     && nc_get_att_float(ncid, varid, "scale_factor",
03878                         &scalfac) == NC_NOERR) {
03879
03880     /* Allocate... */
03881     short *help;
03882     ALLOC(help, short,
03883          EX * EY * EP);
03884
03885     /* Read fill value and missing value... */
03886     short fillval, missval;
03887     if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03888         fillval = 0;
03889     if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03890         missval = 0;
03891
03892     /* Write info... */
03893     LOG(2, "Read 2-D variable: %s"
03894         " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03895         varsel, fillval, missval, scalfac, offset);
03896
03897     /* Read data... */
03898     NC(nc_get_var_short(ncid, varid, help));
03899
03900     /* Copy and check data... */
03901 #pragma omp parallel for default(shared) num_threads(12)
03902     for (int ix = 0; ix < met->nx; ix++)
03903         for (int iy = 0; iy < met->ny; iy++) {
03904             if (init)
03905                 dest[ix][iy] = 0;
03906             short aux = help[ARRAY_2D(iy, ix, met->nx)];
03907             if ((fillval == 0 || aux != fillval)
03908                 && (missval == 0 || aux != missval)
03909                 && fabsf(aux * scalfac + offset) < 1e14f)
03910                 dest[ix][iy] += scl * (aux * scalfac + offset);
03911             else
03912                 dest[ix][iy] = GSL_NAN;
03913         }
03914
03915     /* Free... */
03916     free(help);
03917 }
03918
03919 /* Unpacked data... */
03920 else {
03921
03922     /* Allocate... */
03923     float *help;
03924     ALLOC(help, float,
03925          EX * EY);
03926
03927     /* Read fill value and missing value... */
03928     float fillval, missval;
03929     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03930         fillval = 0;
03931     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03932         missval = 0;
03933
03934     /* Write info... */
03935     LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03936         varsel, fillval, missval);
03937
03938     /* Read data... */
03939     NC(nc_get_var_float(ncid, varid, help));
03940
03941     /* Copy and check data... */
03942 #pragma omp parallel for default(shared) num_threads(12)
03943     for (int ix = 0; ix < met->nx; ix++)
03944         for (int iy = 0; iy < met->ny; iy++) {
03945             if (init)
03946                 dest[ix][iy] = 0;
03947             float aux = help[ARRAY_2D(iy, ix, met->nx)];
03948             if ((fillval == 0 || aux != fillval)
03949                 && (missval == 0 || aux != missval)
03950                 && fabsf(aux) < 1e14f)
03951                 dest[ix][iy] += scl * aux;
03952             else
03953                 dest[ix][iy] = GSL_NAN;
03954         }
03955
03956     /* Free... */

```

```

03957     free(help);
03958 }
03959
03960 /* Return... */
03961 return 1;
03962 }

```

5.21.2.50 read_met_nc_3d() int read_met_nc_3d (

```

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

```

Read and convert 3D variable from meteo data file.

Definition at line 3966 of file libtrac.c.

```

03974     {
03975
03976     char varsel[LEN];
03977
03978     float offset, scalfac;
03979
03980     int varid;
03981
03982     /* Check if variable exists... */
03983     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03984         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03985             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03986             return 0;
03987         } else {
03988             sprintf(varsel, "%s", varname2);
03989         } else
03990             sprintf(varsel, "%s", varname);
03991
03992     /* Read packed data... */
03993     if (ctl->met_nc_scale
03994         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03995         && nc_get_att_float(ncid, varid, "scale_factor",
03996                             &scalfac) == NC_NOERR) {
03997
03998         /* Allocate... */
03999         short *help;
04000         ALLOC(help, short,
04001              EX * EY * EP);
04002
04003         /* Read fill value and missing value... */
04004         short fillval, missval;
04005         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
04006             fillval = 0;
04007         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
04008             missval = 0;
04009
04010         /* Write info... */
04011         LOG(2, "Read 3-D variable: %s "
04012             "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
04013            varsel, fillval, missval, scalfac, offset);
04014
04015         /* Read data... */
04016         NC(nc_get_var_short(ncid, varid, help));
04017
04018         /* Copy and check data... */
04019 #pragma omp parallel for default(shared) num_threads(12)
04020         for (int ix = 0; ix < met->nx; ix++)
04021             for (int iy = 0; iy < met->ny; iy++)
04022                 for (int ip = 0; ip < met->np; ip++) {
04023                     if (init)
04024                         dest[ix][iy][ip] = 0;
04025                     short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04026                     if ((fillval == 0 || aux != fillval)
04027                         && (missval == 0 || aux != missval)
04028                         && fabsf(aux * scalfac + offset) < 1e14f)

```

```

04029         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
04030     else
04031         dest[ix][iy][ip] = GSL_NAN;
04032     }
04033
04034     /* Free... */
04035     free(help);
04036 }
04037
04038 /* Unpacked data... */
04039 else {
04040
04041     /* Allocate... */
04042     float *help;
04043     ALLOC(help, float,
04044           EX * EY * EP);
04045
04046     /* Read fill value and missing value... */
04047     float fillval, missval;
04048     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
04049         fillval = 0;
04050     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
04051         missval = 0;
04052
04053     /* Write info... */
04054     LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
04055         varsel, fillval, missval);
04056
04057     /* Read data... */
04058     NC(nc_get_var_float(ncid, varid, help));
04059
04060     /* Copy and check data... */
04061     #pragma omp parallel for default(shared) num_threads(12)
04062     for (int ix = 0; ix < met->nx; ix++)
04063         for (int iy = 0; iy < met->ny; iy++)
04064             for (int ip = 0; ip < met->np; ip++) {
04065                 if (init)
04066                     dest[ix][iy][ip] = 0;
04067                 float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04068                 if ((fillval == 0 || aux != fillval)
04069                     && (missval == 0 || aux != missval)
04070                     && fabsf(aux) < 1e14f)
04071                     dest[ix][iy][ip] += scl * aux;
04072                 else
04073                     dest[ix][iy][ip] = GSL_NAN;
04074             }
04075
04076     /* Free... */
04077     free(help);
04078 }
04079
04080 /* Return... */
04081 return 1;
04082 }

```

5.21.251 read_met_pbl() void read_met_pbl (
 met_t * met)

Calculate pressure of the boundary layer.

Definition at line 4086 of file libtrac.c.

```

04087     {
04088
04089     /* Set timer... */
04090     SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
04091     LOG(2, "Calculate planetary boundary layer...");
04092
04093     /* Parameters used to estimate the height of the PBL
04094        (e.g., Vogelesang and Holtslag, 1996; Seidel et al., 2012)... */
04095     const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
04096
04097     /* Loop over grid points... */
04098     #pragma omp parallel for default(shared) collapse(2)
04099     for (int ix = 0; ix < met->nx; ix++)
04100         for (int iy = 0; iy < met->ny; iy++) {
04101
04102         /* Set bottom level of PBL... */
04103         double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
04104

```

```

04105      /* Find lowest level near the bottom... */
04106      int ip;
04107      for (ip = 1; ip < met->np; ip++)
04108          if (met->p[ip] < pbl_bot)
04109              break;
04110
04111      /* Get near surface data... */
04112      double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
04113                     met->p[ip], met->z[ix][iy][ip], pbl_bot);
04114      double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
04115                     met->p[ip], met->t[ix][iy][ip], pbl_bot);
04116      double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
04117                     met->p[ip], met->u[ix][iy][ip], pbl_bot);
04118      double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
04119                     met->p[ip], met->v[ix][iy][ip], pbl_bot);
04120      double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
04121                       met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
04122      double tvs = THETA_VIRT(pbl_bot, ts, h2os);
04123
04124      /* Init... */
04125      double rib_old = 0;
04126
04127      /* Loop over levels... */
04128      for (; ip < met->np; ip++) {
04129
04130          /* Get squared horizontal wind speed... */
04131          double vh2
04132              = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
04133          vh2 = GSL_MAX(vh2, SQR(umin));
04134
04135          /* Calculate bulk Richardson number... */
04136          double rib = GO * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
04137              * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
04138                           met->h2o[ix][iy][ip]) - tvs) / vh2;
04139
04140          /* Check for critical value... */
04141          if (rib >= rib_crit) {
04142              met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
04143                                             rib, met->p[ip], rib_crit));
04144              if (met->pbl[ix][iy] > pbl_bot)
04145                  met->pbl[ix][iy] = (float) pbl_bot;
04146              break;
04147          }
04148
04149          /* Save Richardson number... */
04150          rib_old = rib;
04151      }
04152  }
04153 }

```

5.21.2.52 read_met_periodic() void read_met_periodic (
 met_t * met)

Create meteo data with periodic boundary conditions.

Definition at line 4157 of file libtrac.c.

```

04158      {
04159
04160          /* Set timer... */
04161          SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
04162          LOG(2, "Apply periodic boundary conditions...");
04163
04164          /* Check longitudes... */
04165          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
04166                  + met->lon[1] - met->lon[0] - 360) < 0.01))
04167              return;
04168
04169          /* Increase longitude counter... */
04170          if ((++met->nx) > EX)
04171              ERRMSG("Cannot create periodic boundary conditions!");
04172
04173          /* Set longitude... */
04174          met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
04175
04176          /* Loop over latitudes and pressure levels... */
04177          #pragma omp parallel for default(shared)
04178          for (int iy = 0; iy < met->ny; iy++) {
04179              met->ps[met->nx - 1][iy] = met->ps[0][iy];
04180              met->zs[met->nx - 1][iy] = met->zs[0][iy];

```

```

04181     met->ts[met->nx - 1][iy] = met->ts[0][iy];
04182     met->us[met->nx - 1][iy] = met->us[0][iy];
04183     met->vs[met->nx - 1][iy] = met->vs[0][iy];
04184     for (int ip = 0; ip < met->np; ip++) {
04185         met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
04186         met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
04187         met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
04188         met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
04189         met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
04190         met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
04191         met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
04192         met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
04193     }
04194 }
04195 }

```

5.21.2.53 read_met_pv() void read_met_pv (
 met_t * met)

Calculate potential vorticity.

Definition at line 4199 of file libtrac.c.

```

04200     {
04201
04202         double pows[EP];
04203
04204         /* Set timer... */
04205         SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
04206         LOG(2, "Calculate potential vorticity...");
04207
04208         /* Set powers... */
04209         #pragma omp parallel for default(shared)
04210         for (int ip = 0; ip < met->np; ip++)
04211             pows[ip] = pow(1000. / met->p[ip], 0.286);
04212
04213         /* Loop over grid points... */
04214         #pragma omp parallel for default(shared)
04215         for (int ix = 0; ix < met->nx; ix++) {
04216
04217             /* Set indices... */
04218             int ix0 = GSL_MAX(ix - 1, 0);
04219             int ix1 = GSL_MIN(ix + 1, met->nx - 1);
04220
04221             /* Loop over grid points... */
04222             for (int iy = 0; iy < met->ny; iy++) {
04223
04224                 /* Set indices... */
04225                 int iy0 = GSL_MAX(iy - 1, 0);
04226                 int iy1 = GSL_MIN(iy + 1, met->ny - 1);
04227
04228                 /* Set auxiliary variables... */
04229                 double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
04230                 double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
04231                 double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
04232                 double c0 = cos(met->lat[iy0] / 180. * M_PI);
04233                 double c1 = cos(met->lat[iy1] / 180. * M_PI);
04234                 double cr = cos(latr / 180. * M_PI);
04235                 double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
04236
04237                 /* Loop over grid points... */
04238                 for (int ip = 0; ip < met->np; ip++) {
04239
04240                     /* Get gradients in longitude... */
04241                     double dtdx
04242                         = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
04243                     double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
04244
04245                     /* Get gradients in latitude... */
04246                     double dtdy
04247                         = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
04248                     double dudx
04249                         = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
04250
04251                     /* Set indices... */
04252                     int ip0 = GSL_MAX(ip - 1, 0);
04253                     int ip1 = GSL_MIN(ip + 1, met->np - 1);
04254
04255                     /* Get gradients in pressure... */
04256                     double dtdp, dudp, dvdp;

```

```

04257     double dp0 = 100. * (met->p[ip] - met->p[ip0]);
04258     double dp1 = 100. * (met->p[ip1] - met->p[ip]);
04259     if (ip != ip0 && ip != ip1) {
04260         double denom = dp0 * dp1 * (dp0 + dp1);
04261         dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
04262             - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
04263             + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
04264             / denom;
04265         dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
04266             - dp1 * dp1 * met->u[ix][iy][ip0]
04267             + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
04268             / denom;
04269         dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
04270             - dp1 * dp1 * met->v[ix][iy][ip0]
04271             + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
04272             / denom;
04273     } else {
04274         double denom = dp0 + dp1;
04275         dtdp =
04276             (met->t[ix][iy][ip1] * pows[ip1] -
04277             met->t[ix][iy][ip0] * pows[ip0]) / denom;
04278         dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
04279         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
04280     }
04281
04282     /* Calculate PV... */
04283     met->pv[ix][iy][ip] = (float)
04284         (1e6 * G0 *
04285         (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
04286 }
04287 }
04288 }
04289
04290 /* Fix for polar regions... */
04291 #pragma omp parallel for default(shared)
04292 for (int ix = 0; ix < met->nx; ix++)
04293     for (int ip = 0; ip < met->np; ip++) {
04294         met->pv[ix][0][ip]
04295             = met->pv[ix][1][ip]
04296             = met->pv[ix][2][ip];
04297         met->pv[ix][met->ny - 1][ip]
04298             = met->pv[ix][met->ny - 2][ip]
04299             = met->pv[ix][met->ny - 3][ip];
04300     }
04301 }

```

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

Downsampling of meteo data.

Definition at line 4305 of file libtrac.c.

```

04307     {
04308
04309     met_t *help;
04310
04311     /* Check parameters... */
04312     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
04313         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
04314         return;
04315
04316     /* Set timer... */
04317     SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
04318     LOG(2, "Downsampling of meteo data...");
04319
04320     /* Allocate... */
04321     ALLOC(help, met_t, 1);
04322
04323     /* Copy data... */
04324     help->nx = met->nx;
04325     help->ny = met->ny;
04326     help->np = met->np;
04327     memcpy(help->lon, met->lon, sizeof(met->lon));
04328     memcpy(help->lat, met->lat, sizeof(met->lat));
04329     memcpy(help->p, met->p, sizeof(met->p));
04330
04331     /* Smoothing... */

```



```

04332     for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
04333         for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
04334             for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
04335                 help->ps[ix][iy] = 0;
04336                 help->zs[ix][iy] = 0;
04337                 help->ts[ix][iy] = 0;
04338                 help->us[ix][iy] = 0;
04339                 help->vs[ix][iy] = 0;
04340                 help->t[ix][iy][ip] = 0;
04341                 help->u[ix][iy][ip] = 0;
04342                 help->v[ix][iy][ip] = 0;
04343                 help->w[ix][iy][ip] = 0;
04344                 help->h2o[ix][iy][ip] = 0;
04345                 help->o3[ix][iy][ip] = 0;
04346                 help->lwc[ix][iy][ip] = 0;
04347                 help->iwc[ix][iy][ip] = 0;
04348                 float wsum = 0;
04349                 for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
04350                     ix2++) {
04351                     int ix3 = ix2;
04352                     if (ix3 < 0)
04353                         ix3 += met->nx;
04354                     else if (ix3 >= met->nx)
04355                         ix3 -= met->nx;
04356
04357                     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
04358                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
04359                         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
04360                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
04361                             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
04362                                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
04363                                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
04364                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
04365                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
04366                             help->ts[ix][iy] += w * met->ts[ix3][iy2];
04367                             help->us[ix][iy] += w * met->us[ix3][iy2];
04368                             help->vs[ix][iy] += w * met->vs[ix3][iy2];
04369                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
04370                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
04371                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
04372                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
04373                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
04374                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
04375                             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
04376                             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
04377                             wsum += w;
04378                         }
04379                 }
04380                 help->ps[ix][iy] /= wsum;
04381                 help->zs[ix][iy] /= wsum;
04382                 help->ts[ix][iy] /= wsum;
04383                 help->us[ix][iy] /= wsum;
04384                 help->vs[ix][iy] /= wsum;
04385                 help->t[ix][iy][ip] /= wsum;
04386                 help->u[ix][iy][ip] /= wsum;
04387                 help->v[ix][iy][ip] /= wsum;
04388                 help->w[ix][iy][ip] /= wsum;
04389                 help->h2o[ix][iy][ip] /= wsum;
04390                 help->o3[ix][iy][ip] /= wsum;
04391                 help->lwc[ix][iy][ip] /= wsum;
04392                 help->iwc[ix][iy][ip] /= wsum;
04393             }
04394         }
04395     }
04396
04397     /* Downsampling... */
04398     met->nx = 0;
04399     for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04400         met->lon[met->nx] = help->lon[ix];
04401         met->ny = 0;
04402         for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04403             met->lat[met->ny] = help->lat[iy];
04404             met->ps[met->nx][met->ny] = help->ps[ix][iy];
04405             met->zs[met->nx][met->ny] = help->zs[ix][iy];
04406             met->ts[met->nx][met->ny] = help->ts[ix][iy];
04407             met->us[met->nx][met->ny] = help->us[ix][iy];
04408             met->vs[met->nx][met->ny] = help->vs[ix][iy];
04409             met->np = 0;
04410             for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04411                 met->p[met->np] = help->p[ip];
04412                 met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04413                 met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04414                 met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04415                 met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04416                 met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04417                 met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04418                 met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];

```

```

04419         met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04420         met->np++;
04421     }
04422     met->ny++;
04423 }
04424     met->nx++;
04425 }
04426
04427 /* Free... */
04428 free(help);
04429 }

```

5.21.2.55 read_met_surface() void read_met_surface (

```

    int ncid,
    met_t * met,
    ctl_t * ctl )

```

Read surface data.

Definition at line 4433 of file libtrac.c.

```

04436     {
04437
04438     /* Set timer... */
04439     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04440     LOG(2, "Read surface data...");
04441
04442     /* MPTRAC meteo data... */
04443     if (ctl->clams_meteo_data == 0) {
04444
04445         /* Read surface pressure... */
04446         if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
04447             if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04448                 if (!read_met_nc_2d(ncid, "sp", "SP", ctl, met, met->ps, 0.01f, 1)) {
04449                     WARN("Cannot not read surface pressure data (use lowest level)!");
04450                     for (int ix = 0; ix < met->nx; ix++)
04451                         for (int iy = 0; iy < met->ny; iy++)
04452                             met->ps[ix][iy] = (float) met->p[0];
04453                 }
04454             }
04455         } else
04456             for (int ix = 0; ix < met->nx; ix++)
04457                 for (int iy = 0; iy < met->ny; iy++)
04458                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04459
04460         /* Read geopotential height at the surface... */
04461         if (!read_met_nc_2d
04462             (ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
04463             if (!read_met_nc_2d
04464                 (ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
04465                 WARN("Cannot read surface geopotential height!");
04466
04467         /* Read temperature at the surface... */
04468         if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
04469             WARN("Cannot read surface temperature!");
04470
04471         /* Read zonal wind at the surface... */
04472         if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
04473             WARN("Cannot read surface zonal wind!");
04474
04475         /* Read meridional wind at the surface... */
04476         if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
04477             WARN("Cannot read surface meridional wind!");
04478     }
04479
04480     /* CLaMS meteo data... */
04481     else {
04482
04483         /* Read surface pressure... */
04484         if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04485             WARN("Cannot not read surface pressure data (use lowest level)!");
04486             for (int ix = 0; ix < met->nx; ix++)
04487                 for (int iy = 0; iy < met->ny; iy++)
04488                     met->ps[ix][iy] = (float) met->p[0];
04489         }
04490
04491         /* Read geopotential height at the surface
04492            (use lowermost level of 3-D data field)... */
04493         float *help;

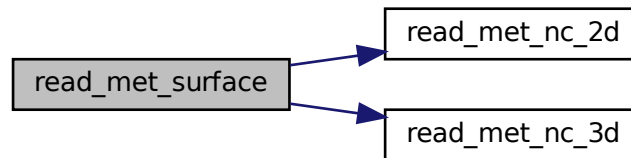
```

```

04494     ALLOC(help, float,
04495           EX * EY * EP);
04496     memcpy(help, met->p1, sizeof(met->p1));
04497     if (!read_met_nc_3d
04498         (ncid, "gph", "GPH", ctl, met, met->p1, (float) (1e-3 / G0), 1)) {
04499         ERRMSG("Cannot read geopotential height!");
04500     } else
04501         for (int ix = 0; ix < met->nx; ix++)
04502             for (int iy = 0; iy < met->ny; iy++)
04503                 met->zs[ix][iy] = met->p1[ix][iy][0];
04504     memcpy(met->p1, help, sizeof(met->p1));
04505     free(help);
04506
04507     /* Read temperature at the surface... */
04508     if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
04509         WARN("Cannot read surface temperature!");
04510
04511     /* Read zonal wind at the surface... */
04512     if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
04513         WARN("Cannot read surface zonal wind!");
04514
04515     /* Read meridional wind at the surface... */
04516     if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
04517         WARN("Cannot read surface meridional wind!");
04518 }
04519 }

```

Here is the call graph for this function:



5.21.2.56 read_met_tropo() void read_met_tropo (

```

    ctl_t * ctl,
    clim_t * clim,
    met_t * met )

```

Calculate tropopause data.

Definition at line 4523 of file libtrac.c.

```

04526     {
04527
04528     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04529           th2[200], z[EP], z2[200];
04530
04531     /* Set timer... */
04532     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04533     LOG(2, "Calculate tropopause...");
04534
04535     /* Get altitude and pressure profiles... */
04536     #pragma omp parallel for default(shared)
04537     for (int iz = 0; iz < met->np; iz++)
04538         z[iz] = Z(met->p[iz]);
04539     #pragma omp parallel for default(shared)
04540     for (int iz = 0; iz <= 190; iz++) {
04541         z2[iz] = 4.5 + 0.1 * iz;
04542         p2[iz] = P(z2[iz]);
04543     }

```

```

04544
04545  /* Do not calculate tropopause... */
04546  if (ctl->met_tropo == 0)
04547 #pragma omp parallel for default(shared) collapse(2)
04548    for (int ix = 0; ix < met->nx; ix++)
04549      for (int iy = 0; iy < met->ny; iy++)
04550        met->pt[ix][iy] = GSL_NAN;
04551
04552  /* Use tropopause climatology... */
04553  else if (ctl->met_tropo == 1) {
04554 #pragma omp parallel for default(shared) collapse(2)
04555    for (int ix = 0; ix < met->nx; ix++)
04556      for (int iy = 0; iy < met->ny; iy++)
04557        met->pt[ix][iy] = (float) clim_tropo(clim, met->time, met->lat[iy]);
04558  }
04559
04560  /* Use cold point... */
04561  else if (ctl->met_tropo == 2) {
04562
04563    /* Loop over grid points... */
04564 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04565    for (int ix = 0; ix < met->nx; ix++)
04566      for (int iy = 0; iy < met->ny; iy++) {
04567
04568        /* Interpolate temperature profile... */
04569        for (int iz = 0; iz < met->np; iz++)
04570          t[iz] = met->t[ix][iy][iz];
04571        spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04572
04573        /* Find minimum... */
04574        int iz = (int) gsl_stats_min_index(t2, 1, 171);
04575        if (iz > 0 && iz < 170)
04576          met->pt[ix][iy] = (float) p2[iz];
04577        else
04578          met->pt[ix][iy] = GSL_NAN;
04579      }
04580  }
04581
04582  /* Use WMO definition... */
04583  else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04584
04585    /* Loop over grid points... */
04586 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04587    for (int ix = 0; ix < met->nx; ix++)
04588      for (int iy = 0; iy < met->ny; iy++) {
04589
04590        /* Interpolate temperature profile... */
04591        int iz;
04592        for (iz = 0; iz < met->np; iz++)
04593          t[iz] = met->t[ix][iy][iz];
04594        spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04595
04596        /* Find 1st tropopause... */
04597        met->pt[ix][iy] = GSL_NAN;
04598        for (iz = 0; iz <= 170; iz++) {
04599          int found = 1;
04600          for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04601            if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04602                ctl->met_tropo_lapse) {
04603              found = 0;
04604              break;
04605            }
04606          if (found) {
04607            if (iz > 0 && iz < 170)
04608              met->pt[ix][iy] = (float) p2[iz];
04609            break;
04610          }
04611        }
04612
04613        /* Find 2nd tropopause... */
04614        if (ctl->met_tropo == 4) {
04615          met->pt[ix][iy] = GSL_NAN;
04616          for (; iz <= 170; iz++) {
04617            int found = 1;
04618            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04619              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04620                  ctl->met_tropo_lapse_sep) {
04621                found = 0;
04622                break;
04623              }
04624            if (found)
04625              break;
04626          }
04627          for (; iz <= 170; iz++) {
04628            int found = 1;
04629            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04630              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >

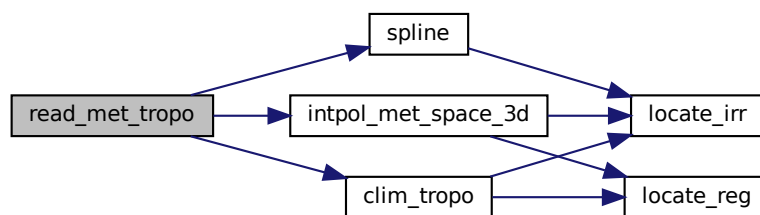
```

```

04631         ctl->met_tropo_lapse) {
04632             found = 0;
04633             break;
04634         }
04635         if (found) {
04636             if (iz > 0 && iz < 170)
04637                 met->pt[ix][iy] = (float) p2[iz];
04638             break;
04639         }
04640     }
04641 }
04642 }
04643 }
04644
04645 /* Use dynamical tropopause... */
04646 else if (ctl->met_tropo == 5) {
04647     /* Loop over grid points... */
04648     #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04649     for (int ix = 0; ix < met->nx; ix++)
04650         for (int iy = 0; iy < met->ny; iy++) {
04651             /* Interpolate potential vorticity profile... */
04652             for (int iz = 0; iz < met->np; iz++)
04653                 pv[iz] = met->pv[ix][iy][iz];
04654             spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04655
04656             /* Interpolate potential temperature profile... */
04657             for (int iz = 0; iz < met->np; iz++)
04658                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04659             spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04660
04661             /* Find dynamical tropopause... */
04662             met->pt[ix][iy] = GSL_NAN;
04663             for (int iz = 0; iz <= 170; iz++)
04664                 if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04665                     || th2[iz] >= ctl->met_tropo_theta) {
04666                     if (iz > 0 && iz < 170)
04667                         met->pt[ix][iy] = (float) p2[iz];
04668                     break;
04669                 }
04670         }
04671     }
04672 }
04673 }
04674
04675 else
04676     ERRMSG("Cannot calculate tropopause!");
04677
04678 /* Interpolate temperature, geopotential height, and water vapor vmr... */
04679 #pragma omp parallel for default(shared) collapse(2)
04680 for (int ix = 0; ix < met->nx; ix++)
04681     for (int iy = 0; iy < met->ny; iy++) {
04682         double h2ot, tt, zt;
04683         INTPOL_INIT;
04684         intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04685                             met->lat[iy], &tt, ci, cw, 1);
04686         intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04687                             met->lat[iy], &z, ci, cw, 0);
04688         intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04689                             met->lat[iy], &h2ot, ci, cw, 0);
04690         met->tt[ix][iy] = (float) tt;
04691         met->z[ix][iy] = (float) z;
04692         met->h2ot[ix][iy] = (float) h2ot;
04693     }
04694 }

```

Here is the call graph for this function:



```

5.21.2.57 read_obs() void read_obs (
    char * filename,
    double * rt,
    double * rz,
    double * rlon,
    double * rlat,
    double * robs,
    int * nobs )

```

Read observation data.

Definition at line 4698 of file `libtrac.c`.

```

04705     {
04706
04707     FILE *in;
04708
04709     char line[LEN];
04710
04711     /* Open observation data file... */
04712     LOG(1, "Read observation data: %s", filename);
04713     if (!(in = fopen(filename, "r")))
04714         ERRMSG("Cannot open file!");
04715
04716     /* Read observations... */
04717     while (fgets(line, LEN, in))
04718         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
04719                 &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
04720             if ((++(*nobs)) >= NOBS)
04721                 ERRMSG("Too many observations!");
04722
04723     /* Close observation data file... */
04724     fclose(in);
04725
04726     /* Check time... */
04727     for (int i = 1; i < *nobs; i++)
04728         if (rt[i] < rt[i - 1])
04729             ERRMSG("Time must be ascending!");
04730
04731     /* Write info... */
04732     int n = *nobs;
04733     double mini, maxi;
04734     LOG(2, "Number of observations: %d", *nobs);
04735     gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
04736     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04737     gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
04738     LOG(2, "Altitude range: %g ... %g km", mini, maxi);
04739     gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
04740     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04741     gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
04742     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04743     gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
04744     LOG(2, "Observation range: %g ... %g", mini, maxi);
04745 }

```

```

5.21.2.58 scan_ctl() double scan_ctl (
    const char * filename,
    int argc,
    char * argv[],
    const char * varname,
    int arridx,
    const char * defvalue,
    char * value )

```

Read a control parameter from file or command line.

Definition at line 4749 of file `libtrac.c`.

```

04756         {
04757
04758     FILE *in = NULL;
04759
04760     char fullname1[LEN], fullname2[LEN], rval[LEN];
04761
04762     int contain = 0, i;
04763
04764     /* Open file... */
04765     if (filename[strlen(filename) - 1] != '-')
04766         if (!(in = fopen(filename, "r")))
04767             ERRMSG("Cannot open file!");
04768
04769     /* Set full variable name... */
04770     if (arridx >= 0) {
04771         sprintf(fullname1, "%s[%d]", varname, arridx);
04772         sprintf(fullname2, "%s[*]", varname);
04773     } else {
04774         sprintf(fullname1, "%s", varname);
04775         sprintf(fullname2, "%s", varname);
04776     }
04777
04778     /* Read data... */
04779     if (in != NULL) {
04780         char dummy[LEN], line[LEN], rvarname[LEN];
04781         while (fgets(line, LEN, in)) {
04782             if (sscanf(line, "%4999s %4999s", rvarname, dummy, rval) == 3)
04783                 if (strcmp(rvarname, fullname1) == 0 ||
04784                     strcmp(rvarname, fullname2) == 0) {
04785                     contain = 1;
04786                     break;
04787                 }
04788         }
04789     }
04790     for (i = 1; i < argc - 1; i++)
04791         if (strcmp(argv[i], fullname1) == 0 ||
04792             strcmp(argv[i], fullname2) == 0) {
04793             sprintf(rval, "%s", argv[i + 1]);
04794             contain = 1;
04795             break;
04796         }
04797
04798     /* Close file... */
04799     if (in != NULL)
04800         fclose(in);
04801
04802     /* Check for missing variables... */
04803     if (!contain) {
04804         if (strlen(defvalue) > 0)
04805             sprintf(rval, "%s", defvalue);
04806         else
04807             ERRMSG("Missing variable %s!\n", fullname1);
04808     }
04809
04810     /* Write info... */
04811     LOG(1, "%s = %s", fullname1, rval);
04812
04813     /* Return values... */
04814     if (value != NULL)
04815         sprintf(value, "%s", rval);
04816     return atof(rval);
04817 }

```

5.21.2.59 sedi() double sedi (
double p,
double T,
double rp,
double rhop)

Calculate sedimentation velocity.

Definition at line 4821 of file libtrac.c.

```

04825     {
04826
04827     /* Convert particle radius from microns to m... */
04828     rp *= 1e-6;
04829

```

```

04830  /* Density of dry air [kg / m^3]... */
04831  double rho = RHO(p, T);
04832
04833  /* Dynamic viscosity of air [kg / (m s)]... */
04834  double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04835
04836  /* Thermal velocity of an air molecule [m / s]... */
04837  double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04838
04839  /* Mean free path of an air molecule [m]... */
04840  double lambda = 2. * eta / (rho * v);
04841
04842  /* Knudsen number for air (dimensionless)... */
04843  double K = lambda / rp;
04844
04845  /* Cunningham slip-flow correction (dimensionless)... */
04846  double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04847
04848  /* Sedimentation velocity [m / s]... */
04849  return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
04850 }

```

5.21.2.60 spline() void spline (

```

    double * x,
    double * y,
    int n,
    double * x2,
    double * y2,
    int n2,
    int method )

```

Spline interpolation.

Definition at line 4854 of file libtrac.c.

```

04861  {
04862
04863  /* Cubic spline interpolation... */
04864  if (method == 1) {
04865
04866      /* Allocate... */
04867      gsl_interp_accel *acc;
04868      gsl_spline *s;
04869      acc = gsl_interp_accel_alloc();
04870      s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04871
04872      /* Interpolate profile... */
04873      gsl_spline_init(s, x, y, (size_t) n);
04874      for (int i = 0; i < n2; i++)
04875          if (x2[i] <= x[0])
04876              y2[i] = y[0];
04877          else if (x2[i] >= x[n - 1])
04878              y2[i] = y[n - 1];
04879          else
04880              y2[i] = gsl_spline_eval(s, x2[i], acc);
04881
04882      /* Free... */
04883      gsl_spline_free(s);
04884      gsl_interp_accel_free(acc);
04885  }
04886
04887  /* Linear interpolation... */
04888  else {
04889      for (int i = 0; i < n2; i++)
04890          if (x2[i] <= x[0])
04891              y2[i] = y[0];
04892          else if (x2[i] >= x[n - 1])
04893              y2[i] = y[n - 1];
04894          else {
04895              int idx = locate_irr(x, n, x2[i]);
04896              y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04897          }
04898  }
04899  }

```


Here is the call graph for this function:



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

Calculate standard deviation.

Definition at line 4903 of file libtrac.c.

```

04905     {
04906
04907     if (n <= 0)
04908         return 0;
04909
04910     float mean = 0, var = 0;
04911
04912     for (int i = 0; i < n; ++i) {
04913         mean += data[i];
04914         var += SQR(data[i]);
04915     }
04916
04917     var = var / (float) n - SQR(mean / (float) n);
04918
04919     return (var > 0 ? sqrtf(var) : 0);
04920 }
  
```

5.21.2.62 sza() double sza (
 double sec,
 double lon,
 double lat)

Calculate solar zenith angle.

Definition at line 4924 of file libtrac.c.

```

04927     {
04928
04929     double D, dec, e, g, GMST, h, L, LST, q, ra;
04930
04931     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
04932     D = sec / 86400 - 0.5;
04933
04934     /* Geocentric apparent ecliptic longitude [rad]... */
04935     g = (357.529 + 0.98560028 * D) * M_PI / 180;
04936     q = 280.459 + 0.98564736 * D;
04937     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
04938
04939     /* Mean obliquity of the ecliptic [rad]... */
04940     e = (23.439 - 0.00000036 * D) * M_PI / 180;
04941
04942     /* Declination [rad]... */
04943     dec = asin(sin(e) * sin(L));
04944 }
  
```

```

04945  /* Right ascension [rad]... */
04946  ra = atan2(cos(e) * sin(L), cos(L));
04947
04948  /* Greenwich Mean Sidereal Time [h]... */
04949  GMST = 18.697374558 + 24.06570982441908 * D;
04950
04951  /* Local Sidereal Time [h]... */
04952  LST = GMST + lon / 15;
04953
04954  /* Hour angle [rad]... */
04955  h = LST / 12 * M_PI - ra;
04956
04957  /* Convert latitude... */
04958  lat *= M_PI / 180;
04959
04960  /* Return solar zenith angle [rad]... */
04961  return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
04962 }

```

5.21.2.63 time2jsec() void time2jsec (

```

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

```

Convert date to seconds.

Definition at line 4966 of file libtrac.c.

```

04974  {
04975
04976  struct tm t0, t1;
04977
04978  t0.tm_year = 100;
04979  t0.tm_mon = 0;
04980  t0.tm_mday = 1;
04981  t0.tm_hour = 0;
04982  t0.tm_min = 0;
04983  t0.tm_sec = 0;
04984
04985  t1.tm_year = year - 1900;
04986  t1.tm_mon = mon - 1;
04987  t1.tm_mday = day;
04988  t1.tm_hour = hour;
04989  t1.tm_min = min;
04990  t1.tm_sec = sec;
04991
04992  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04993 }

```

5.21.2.64 timer() void timer (

```

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

```

Measure wall-clock time.

Definition at line 4997 of file libtrac.c.

```

05000  {
05001
05002  static char names[NTIMER][100], groups[NTIMER][100];
05003
05004  static double rt_name[NTIMER], rt_group[NTIMER],

```

```

05005     rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;
05006
05007     static int iname = -1, igrp = -1, nname, ngroup, ct_name[NTIMER];
05008
05009     /* Get time... */
05010     t1 = omp_get_wtime();
05011     dt = t1 - t0;
05012
05013     /* Add elapsed time to current timers... */
05014     if (iname >= 0) {
05015         rt_name[iname] += dt;
05016         rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
05017         rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
05018         ct_name[iname]++;
05019     }
05020     if (igrp >= 0)
05021         rt_group[igrp] += t1 - t0;
05022
05023     /* Report timers... */
05024     if (output) {
05025         for (int i = 0; i < nname; i++)
05026             LOG(1, "TIMER_%s = %.3f s (min= %g s, mean= %g s, "
05027                 " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
05028                 rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
05029         for (int i = 0; i < ngroup; i++)
05030             LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
05031         double total = 0.0;
05032         for (int i = 0; i < nname; i++)
05033             total += rt_name[i];
05034         LOG(1, "TIMER_TOTAL = %.3f s", total);
05035     }
05036
05037     /* Identify IDs of next timer... */
05038     for (iname = 0; iname < nname; iname++)
05039         if (strcmp(name, names[iname]) == 0)
05040             break;
05041     for (igrp = 0; igrp < ngroup; igrp++)
05042         if (strcmp(group, groups[igrp]) == 0)
05043             break;
05044
05045     /* Check whether this is a new timer... */
05046     if (iname >= nname) {
05047         sprintf(names[iname], "%s", name);
05048         if ((++nname) > NTIMER)
05049             ERRMSG("Too many timers!");
05050     }
05051
05052     /* Check whether this is a new group... */
05053     if (igrp >= ngroup) {
05054         sprintf(groups[igrp], "%s", group);
05055         if ((++ngroup) > NTIMER)
05056             ERRMSG("Too many groups!");
05057     }
05058
05059     /* Save starting time... */
05060     t0 = t1;
05061 }

```

5.21.2.65 tropo_weight() double tropo_weight (
 clim_t * clim,
 double t,
 double lat,
 double p)

Get weighting factor based on tropopause distance.

Definition at line 5065 of file libtrac.c.

```

05069     {
05070
05071     /* Get tropopause pressure... */
05072     double pt = clim_tropo(clim, t, lat);
05073
05074     /* Get pressure range... */
05075     double p1 = pt * 0.866877899;
05076     double p0 = pt / 0.866877899;
05077
05078     /* Get weighting factor... */

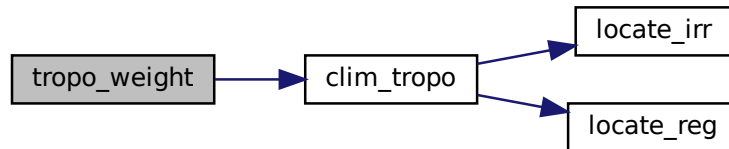
```

```

05079  if (p > p0)
05080      return 1;
05081  else if (p < p1)
05082      return 0;
05083  else
05084      return LIN(p0, 1.0, p1, 0.0, p);
05085  }

```

Here is the call graph for this function:



5.21.2.66 write_atm() void write_atm (

const char * filename,

ctl_t * ctl,

atm_t * atm,

double t)

Write atmospheric data.

Definition at line 5089 of file libtrac.c.

```

05093  {
05094
05095  /* Set timer... */
05096  SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
05097
05098  /* Write info... */
05099  LOG(1, "Write atmospheric data: %s", filename);
05100
05101  /* Write ASCII data... */
05102  if (ctl->atm_type == 0)
05103      write_atm_asc(filename, ctl, atm, t);
05104
05105  /* Write binary data... */
05106  else if (ctl->atm_type == 1)
05107      write_atm_bin(filename, ctl, atm);
05108
05109  /* Write netCDF data... */
05110  else if (ctl->atm_type == 2)
05111      write_atm_nc(filename, ctl, atm);
05112
05113  /* Write CLaMS data... */
05114  else if (ctl->atm_type == 3)
05115      write_atm_clams(ctl, atm, t);
05116
05117  /* Error... */
05118  else
05119      ERRMSG("Atmospheric data type not supported!");
05120
05121  /* Write info... */
05122  double mini, maxi;
05123  LOG(2, "Number of particles: %d", atm->np);
05124  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
05125  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
05126  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
05127  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
05128  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
05129  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);

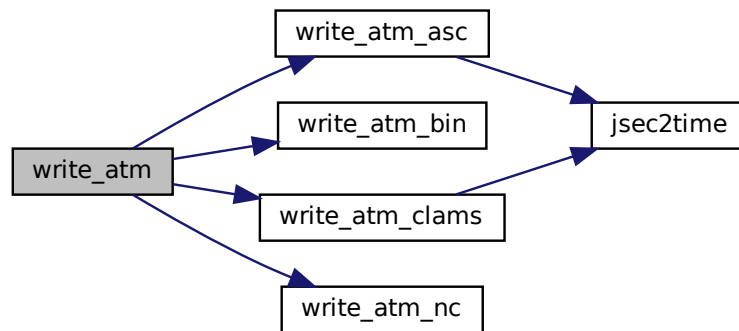
```

```

05130 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
05131 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
05132 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
05133 for (int iq = 0; iq < ctl->nq; iq++) {
05134     char msg[LEN];
05135     sprintf(msg, "Quantity %s range: %s ... %s %s",
05136             ctl->qnt_name[iq], ctl->qnt_format[iq],
05137             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
05138     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
05139     LOG(2, msg, mini, maxi);
05140 }
05141 }

```

Here is the call graph for this function:



5.21.2.67 write_atm_asc() void write_atm_asc (

```

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

```

Write atmospheric data in ASCII format.

Definition at line 5145 of file libtrac.c.

```

05149 {
05150
05151     FILE *out;
05152
05153     /* Set time interval for output... */
05154     double t0 = t - 0.5 * ctl->dt_mod;
05155     double t1 = t + 0.5 * ctl->dt_mod;
05156
05157     /* Check if gnuplot output is requested... */
05158     if (ctl->atm_gpfile[0] != '-') {
05159
05160         /* Create gnuplot pipe... */
05161         if (!(out = popen("gnuplot", "w")))
05162             ERRMSG("Cannot create pipe to gnuplot!");
05163
05164         /* Set plot filename... */
05165         fprintf(out, "set out \"%s.png\"\n", filename);
05166
05167         /* Set time string... */
05168         double r;
05169         int year, mon, day, hour, min, sec;
05170         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05171         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05172                 year, mon, day, hour, min);

```

```

05173
05174     /* Dump gnuplot file to pipe... */
05175     FILE *in;
05176     if (!(in = fopen(ctl->atm_gpfile, "r")))
05177         ERRMSG("Cannot open file!");
05178     char line[LEN];
05179     while (fgets(line, LEN, in))
05180         fprintf(out, "%s", line);
05181     fclose(in);
05182 }
05183
05184 else {
05185     /* Create file... */
05186     if (!(out = fopen(filename, "w")))
05187         ERRMSG("Cannot create file!");
05188 }
05189
05190
05191 /* Write header... */
05192 fprintf(out,
05193         "# $1 = time [s]\n"
05194         "# $2 = altitude [km]\n"
05195         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05196 for (int iq = 0; iq < ctl->nq; iq++)
05197     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
05198             ctl->qnt_unit[iq]);
05199 fprintf(out, "\n");
05200
05201 /* Write data... */
05202 for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
05203
05204     /* Check time... */
05205     if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05206         continue;
05207
05208     /* Write output... */
05209     fprintf(out, "%.2f %g %g %g", atm->time[ip], 2(atm->p[ip]),
05210             atm->lon[ip], atm->lat[ip]);
05211     for (int iq = 0; iq < ctl->nq; iq++) {
05212         fprintf(out, " ");
05213         if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05214             fprintf(out, ctl->qnt_format[iq], GSL_NAN);
05215         else
05216             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05217     }
05218     fprintf(out, "\n");
05219 }
05220
05221 /* Close file... */
05222 fclose(out);
05223 }

```

Here is the call graph for this function:



5.21.2.68 write_atm_bin() void write_atm_bin (
 const char * filename,
 ctl_t * ctl,
 atm_t * atm)

Write atmospheric data in binary format.

Definition at line 5227 of file libtrac.c.

```

05230     {
05231
05232     FILE *out;
05233
05234     /* Create file... */
05235     if (!(out = fopen(filename, "w")))
05236         ERRMSG("Cannot create file!");
05237
05238     /* Write version of binary data... */
05239     int version = 100;
05240     FWRITE(&version, int,
05241           1,
05242           out);
05243
05244     /* Write data... */
05245     FWRITE(&atm->np, int,
05246           1,
05247           out);
05248     FWRITE(atm->time, double,
05249           (size_t) atm->np,
05250           out);
05251     FWRITE(atm->p, double,
05252           (size_t) atm->np,
05253           out);
05254     FWRITE(atm->lon, double,
05255           (size_t) atm->np,
05256           out);
05257     FWRITE(atm->lat, double,
05258           (size_t) atm->np,
05259           out);
05260     for (int iq = 0; iq < ctl->nq; iq++)
05261         FWRITE(atm->q[iq], double,
05262               (size_t) atm->np,
05263               out);
05264
05265     /* Write final flag... */
05266     int final = 999;
05267     FWRITE(&final, int,
05268           1,
05269           out);
05270
05271     /* Close file... */
05272     fclose(out);
05273 }

```

5.21.2.69 write_atm_clams() void write_atm_clams (
 ctl_t * ctl,
 atm_t * atm,
 double t)

Write atmospheric data in CLaMS format.

Definition at line 5277 of file libtrac.c.

```

05280     {
05281
05282     /* Global Counter... */
05283     static size_t out_cnt = 0;
05284
05285     char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";
05286
05287     double r, r_start, r_stop;
05288
05289     int year, mon, day, hour, min, sec;
05290     int year_start, mon_start, day_start, hour_start, min_start, sec_start;
05291     int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
05292     int ncid, varid, tid, cid, zid, dim_ids[2];
05293
05294     /* time, nparc */
05295     size_t start[2], count[2];
05296
05297     /* Determine start and stop times of calculation... */
05298     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05299     jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
05300             &min_start, &sec_start, &r_start);
05301     jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
05302             &min_stop, &sec_stop, &r_stop);
05303 }

```

```

05304  /* Set filename... */
05305  sprintf(filename_out,
05306          "./traj_fix_3d_%02d%02d%02d%02d_%02d%02d%02d%02d.nc",
05307          year_start % 100, mon_start, day_start, hour_start,
05308          year_stop % 100, mon_stop, day_stop, hour_stop);
05309  printf("Write traj file: %s\n", filename_out);
05310
05311  /* Define hyperslap for the traj_file... */
05312  start[0] = out_cnt;
05313  start[1] = 0;
05314  count[0] = 1;
05315  count[1] = (size_t) atm->np;
05316
05317  /* Create the file at the first timestep... */
05318  if (out_cnt == 0) {
05319
05320      /* Create file... */
05321      nc_create(filename_out, NC_CLOBBER, &ncid);
05322
05323      /* Define dimensions... */
05324      NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
05325      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05326      NC(nc_def_dim(ncid, "TMDT", 7, &cid));
05327      dim_ids[0] = tid;
05328      dim_ids[1] = pid;
05329
05330      /* Define variables and their attributes... */
05331      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05332               "seconds since 2000-01-01 00:00:00 UTC");
05333      NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
05334      NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
05335      NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
05336      NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
05337      for (int iq = 0; iq < ctl->nq; iq++)
05338          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05339                   ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05340
05341      /* Define global attributes... */
05342      NC_PUT_ATT_GLOBAL("exp_VERTCOORD_name", "zeta");
05343      NC_PUT_ATT_GLOBAL("model", "MPTRAC");
05344
05345      /* End definitions... */
05346      NC(nc_enddef(ncid));
05347      NC(nc_close(ncid));
05348  }
05349
05350  /* Increment global counter to change hyperslap... */
05351  out_cnt++;
05352
05353  /* Open file... */
05354  NC(nc_open(filename_out, NC_WRITE, &ncid));
05355
05356  /* Write data... */
05357  NC_PUT_DOUBLE("time", atm->time, 1);
05358  NC_PUT_DOUBLE("LAT", atm->lat, 1);
05359  NC_PUT_DOUBLE("LON", atm->lon, 1);
05360  NC_PUT_DOUBLE("PRESS", atm->p, 1);
05361  if (ctl->vert_coord_ap == 1) {
05362      NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
05363  } else if (ctl->qnt_zeta >= 0) {
05364      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
05365  }
05366  for (int iq = 0; iq < ctl->nq; iq++)
05367      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);
05368
05369  /* Close file... */
05370  NC(nc_close(ncid));
05371
05372  /* At the last time step create the init_fix_YYYYMMDDHH file... */
05373  if ((year == year_stop) && (mon == mon_stop)
05374      && (day == day_stop) && (hour == hour_stop)) {
05375
05376      /* Set filename... */
05377      char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
05378      sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
05379              year_stop % 100, mon_stop, day_stop, hour_stop);
05380      printf("Write init file: %s\n", filename_init);
05381
05382      /* Create file... */
05383      nc_create(filename_init, NC_CLOBBER, &ncid);
05384
05385      /* Define dimensions... */
05386      NC(nc_def_dim(ncid, "time", 1, &tid));
05387      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05388      dim_ids[0] = tid;
05389      dim_ids[1] = pid;
05390

```

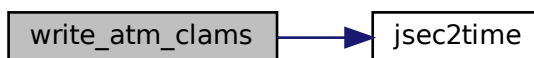


```

05391      /* Define variables and their attributes... */
05392      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05393                "seconds since 2000-01-01 00:00:00 UTC");
05394      NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
05395      NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
05396      NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
05397      NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
05398      NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
05399      NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
05400      for (int iq = 0; iq < ctl->nq; iq++)
05401          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05402                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05403
05404      /* Define global attributes... */
05405      NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
05406      NC_PUT_ATT_GLOBAL("model", "MPTRAC");
05407
05408      /* End definitions... */
05409      NC(nc_enddef(ncid));
05410
05411      /* Write data... */
05412      NC_PUT_DOUBLE("time", atm->time, 0);
05413      NC_PUT_DOUBLE("LAT", atm->lat, 0);
05414      NC_PUT_DOUBLE("LON", atm->lon, 0);
05415      NC_PUT_DOUBLE("PRESS", atm->p, 0);
05416      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
05417      for (int iq = 0; iq < ctl->nq; iq++)
05418          NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05419
05420      /* Close file... */
05421      NC(nc_close(ncid));
05422  }
05423 }

```

Here is the call graph for this function:



5.21.2.70 write_atm_nc() void write_atm_nc (

 const char * filename,

 ctl_t * ctl,

 atm_t * atm)

Write atmospheric data in netCDF format.

Definition at line 5427 of file libtrac.c.

```

05430      {
05431
05432      int ncid, obsid, varid;
05433
05434      size_t start[2], count[2];
05435
05436      /* Create file... */
05437      nc_create(filename, NC_CLOBBER, &ncid);
05438
05439      /* Define dimensions... */
05440      NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));
05441
05442      /* Define variables and their attributes... */
05443      NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
05444                "seconds since 2000-01-01 00:00:00 UTC");
05445      NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");

```

```

05446 NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
05447 NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
05448 for (int iq = 0; iq < ctl->nq; iq++)
05449     NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
05450               ctl->qnt_longname[iq], ctl->qnt_unit[iq]);
05451
05452 /* Define global attributes... */
05453 NC_PUT_ATT_GLOBAL("featureType", "point");
05454
05455 /* End definitions... */
05456 NC(nc_enddef(ncid));
05457
05458 /* Write data... */
05459 NC_PUT_DOUBLE("time", atm->time, 0);
05460 NC_PUT_DOUBLE("press", atm->p, 0);
05461 NC_PUT_DOUBLE("lon", atm->lon, 0);
05462 NC_PUT_DOUBLE("lat", atm->lat, 0);
05463 for (int iq = 0; iq < ctl->nq; iq++)
05464     NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05465
05466 /* Close file... */
05467 NC(nc_close(ncid));
05468 }

```

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

Write CSI data.

Definition at line 5472 of file libtrac.c.

```

05476 {
05477
05478 static FILE *out;
05479
05480 static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
05481             dlon, dlat, dz, x[NCSI], y[NCSI];
05482
05483 static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;
05484
05485 /* Set timer... */
05486 SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
05487
05488 /* Init... */
05489 if (t == ctl->t_start) {
05490
05491     /* Check quantity index for mass... */
05492     if (ctl->qnt_m < 0)
05493         ERRMSG("Need quantity mass!");
05494
05495     /* Allocate... */
05496     ALLOC(area, double,
05497           ctl->csi_ny);
05498     ALLOC(rt, double,
05499           NOBS);
05500     ALLOC(rz, double,
05501           NOBS);
05502     ALLOC(rlon, double,
05503           NOBS);
05504     ALLOC(rlat, double,
05505           NOBS);
05506     ALLOC(robs, double,
05507           NOBS);
05508
05509     /* Read observation data... */
05510     read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);
05511
05512     /* Create new file... */
05513     LOG(1, "Write CSI data: %s", filename);
05514     if (!(out = fopen(filename, "w")))
05515         ERRMSG("Cannot create file!");
05516
05517     /* Write header... */
05518     fprintf(out,
05519            "# $1 = time [s]\n"

```

```

05520         "# $2 = number of hits (cx)\n"
05521         "# $3 = number of misses (cy)\n"
05522         "# $4 = number of false alarms (cz)\n"
05523         "# $5 = number of observations (cx + cy)\n"
05524         "# $6 = number of forecasts (cx + cz)\n"
05525         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
05526         "# $8 = probability of detection (POD) [%%]\n"
05527         "# $9 = false alarm rate (FAR) [%%]\n"
05528         "# $10 = critical success index (CSI) [%%]\n");
05529     fprintf(out,
05530         "# $11 = hits associated with random chance\n"
05531         "# $12 = equitable threat score (ETS) [%%]\n"
05532         "# $13 = Pearson linear correlation coefficient\n"
05533         "# $14 = Spearman rank-order correlation coefficient\n"
05534         "# $15 = column density mean error (F - O) [kg/m^2]\n"
05535         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
05536         "# $17 = column density mean absolute error [kg/m^2]\n"
05537         "# $18 = number of data points\n\n");
05538
05539     /* Set grid box size... */
05540     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
05541     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
05542     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
05543
05544     /* Set horizontal coordinates... */
05545     for (iy = 0; iy < ctl->csi_ny; iy++) {
05546         double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
05547         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
05548     }
05549 }
05550
05551     /* Set time interval... */
05552     double t0 = t - 0.5 * ctl->dt_mod;
05553     double t1 = t + 0.5 * ctl->dt_mod;
05554
05555     /* Allocate... */
05556     ALLOC(modmean, double,
05557         ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05558     ALLOC(obsmean, double,
05559         ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05560     ALLOC(obscount, int,
05561         ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05562
05563     /* Loop over observations... */
05564     for (int i = 0; i < nobs; i++) {
05565
05566         /* Check time... */
05567         if (rt[i] < t0)
05568             continue;
05569         else if (rt[i] >= t1)
05570             break;
05571
05572         /* Check observation data... */
05573         if (!isfinite(robs[i]))
05574             continue;
05575
05576         /* Calculate indices... */
05577         ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
05578         iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
05579         iz = (int) ((rz[i] - ctl->csi_z0) / dz);
05580
05581         /* Check indices... */
05582         if (ix < 0 || ix >= ctl->csi_nx ||
05583             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05584             continue;
05585
05586         /* Get mean observation index... */
05587         int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05588         obsmean[idx] += robs[i];
05589         obscount[idx]++;
05590     }
05591
05592     /* Analyze model data... */
05593     for (ip = 0; ip < atm->np; ip++) {
05594
05595         /* Check time... */
05596         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05597             continue;
05598
05599         /* Get indices... */
05600         ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
05601         iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
05602         iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
05603
05604         /* Check indices... */
05605         if (ix < 0 || ix >= ctl->csi_nx ||
05606             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)

```

```

05607         continue;
05608
05609         /* Get total mass in grid cell... */
05610         int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05611         modmean[idx] += atm->q[ctl->qnt_m][ip];
05612     }
05613
05614     /* Analyze all grid cells... */
05615     for (ix = 0; ix < ctl->csi_nx; ix++)
05616         for (iy = 0; iy < ctl->csi_ny; iy++)
05617             for (iz = 0; iz < ctl->csi_nz; iz++) {
05618
05619                 /* Calculate mean observation index... */
05620                 int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05621                 if (obscount[idx] > 0)
05622                     obsmean[idx] /= obscount[idx];
05623
05624                 /* Calculate column density... */
05625                 if (modmean[idx] > 0)
05626                     modmean[idx] /= (1e6 * area[iy]);
05627
05628                 /* Calculate CSI... */
05629                 if (obscount[idx] > 0) {
05630                     ct++;
05631                     if (obsmean[idx] >= ctl->csi_obsmin &&
05632                         modmean[idx] >= ctl->csi_modmin)
05633                         cx++;
05634                     else if (obsmean[idx] >= ctl->csi_obsmin &&
05635                             modmean[idx] < ctl->csi_modmin)
05636                         cy++;
05637                     else if (obsmean[idx] < ctl->csi_obsmin &&
05638                             modmean[idx] >= ctl->csi_modmin)
05639                         cz++;
05640                 }
05641
05642                 /* Save data for other verification statistics... */
05643                 if (obscount[idx] > 0
05644                     && (obsmean[idx] >= ctl->csi_obsmin
05645                         || modmean[idx] >= ctl->csi_modmin)) {
05646                     x[n] = modmean[idx];
05647                     y[n] = obsmean[idx];
05648                     if ((++n) > NCSI)
05649                         ERRMSG("Too many data points to calculate statistics!");
05650                 }
05651             }
05652
05653     /* Write output... */
05654     if (fmod(t, ctl->csi_dt_out) == 0) {
05655
05656         /* Calculate verification statistics
05657         (https://www.cawcr.gov.au/projects/verification/) ... */
05658         static double work[2 * NCSI];
05659         int n_obs = cx + cy;
05660         int n_for = cx + cz;
05661         double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
05662         double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
05663         double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
05664         double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
05665         double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
05666         double ets = (cx + cy + cz - cx_rd > 0) ?
05667             (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
05668         double rho_p =
05669             (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
05670         double rho_s =
05671             (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
05672         for (int i = 0; i < n; i++)
05673             work[i] = x[i] - y[i];
05674         double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
05675         double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
05676             0.0) : GSL_NAN;
05677         double absdev =
05678             (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
05679
05680         /* Write... */
05681         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %d\n",
05682             t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
05683             rho_p, rho_s, mean, rmse, absdev, n);
05684
05685         /* Set counters to zero... */
05686         n = ct = cx = cy = cz = 0;
05687     }
05688
05689     /* Free... */
05690     free(modmean);
05691     free(obsmean);
05692     free(obscount);
05693

```

```

05694  /* Finalize... */
05695  if (t == ctl->t_stop) {
05696
05697      /* Close output file... */
05698      fclose(out);
05699
05700      /* Free... */
05701      free(area);
05702      free(rt);
05703      free(rz);
05704      free(rlon);
05705      free(rlat);
05706      free(robs);
05707  }
05708 }

```

Here is the call graph for this function:



5.21.2.72 write_ens() void write_ens (

```

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

```

Write ensemble data.

Definition at line 5712 of file libtrac.c.

```

05716  {
05717
05718      static FILE *out;
05719
05720      static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
05721          x[3], zm[NENS];
05722
05723      static int n[NENS];
05724
05725      /* Set timer... */
05726      SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
05727
05728      /* Check quantities... */
05729      if (ctl->qnt_ens < 0)
05730          ERRMSG("Missing ensemble IDs!");
05731
05732      /* Set time interval... */
05733      double t0 = t - 0.5 * ctl->dt_mod;
05734      double t1 = t + 0.5 * ctl->dt_mod;
05735
05736      /* Init... */
05737      for (int i = 0; i < NENS; i++) {
05738          for (int iq = 0; iq < ctl->nq; iq++)
05739              qm[iq][i] = qs[iq][i] = 0;
05740          xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
05741          n[i] = 0;
05742      }
05743
05744      /* Loop over air parcels... */
05745      for (int ip = 0; ip < atm->np; ip++) {
05746
05747          /* Check time... */
05748          if (atm->time[ip] < t0 || atm->time[ip] > t1)

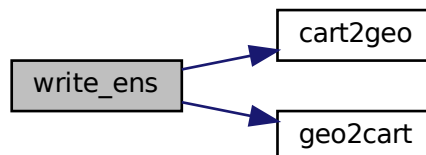
```

```

05749         continue;
05750
05751         /* Check ensemble ID... */
05752         if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
05753             ERRMSG("Ensemble ID is out of range!");
05754
05755         /* Get means... */
05756         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
05757         for (int iq = 0; iq < ctl->nq; iq++) {
05758             qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
05759             qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
05760         }
05761         xm[ctl->qnt_ens][0] += x[0];
05762         xm[ctl->qnt_ens][1] += x[1];
05763         xm[ctl->qnt_ens][2] += x[2];
05764         zm[ctl->qnt_ens] += Z(atm->p[ip]);
05765         n[ctl->qnt_ens]++;
05766     }
05767
05768     /* Create file... */
05769     LOG(1, "Write ensemble data: %s", filename);
05770     if (!out = fopen(filename, "w"))
05771         ERRMSG("Cannot create file!");
05772
05773     /* Write header... */
05774     fprintf(out,
05775             "# $1 = time [s]\n"
05776             "# $2 = altitude [km]\n"
05777             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05778     for (int iq = 0; iq < ctl->nq; iq++)
05779         fprintf(out, "# $qd = %s (mean) [%s]\n", 5 + iq,
05780                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05781     for (int iq = 0; iq < ctl->nq; iq++)
05782         fprintf(out, "# $qd = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
05783                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05784     fprintf(out, "# $qd = number of members\n\n", 5 + 2 * ctl->nq);
05785
05786     /* Write data... */
05787     for (int i = 0; i < NENS; i++)
05788         if (n[i] > 0) {
05789             cart2geo(xm[i], &dummy, &lon, &lat);
05790             fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
05791             for (int iq = 0; iq < ctl->nq; iq++) {
05792                 fprintf(out, " ");
05793                 fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
05794             }
05795             for (int iq = 0; iq < ctl->nq; iq++) {
05796                 fprintf(out, " ");
05797                 double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
05798                 fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
05799             }
05800             fprintf(out, " %d\n", n[i]);
05801         }
05802
05803     /* Close file... */
05804     fclose(out);
05805 }

```

Here is the call graph for this function:



5.21.2.73 write_grid() void write_grid (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write gridded data.

Definition at line 5809 of file libtrac.c.

```

5815     {
5816
5817     double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;
5818
5819     int *ixs, *iys, *izs, *np;
5820
5821     /* Set timer... */
5822     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
5823
5824     /* Write info... */
5825     LOG(1, "Write grid data: %s", filename);
5826
5827     /* Allocate... */
5828     ALLOC(cd, double,
5829          ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5830     ALLOC(mass, double,
5831          ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5832     ALLOC(vmr_expl, double,
5833          ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5834     ALLOC(vmr_impl, double,
5835          ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5836     ALLOC(z, double,
5837          ctl->grid_nz);
5838     ALLOC(lon, double,
5839          ctl->grid_nx);
5840     ALLOC(lat, double,
5841          ctl->grid_ny);
5842     ALLOC(area, double,
5843          ctl->grid_ny);
5844     ALLOC(press, double,
5845          ctl->grid_nz);
5846     ALLOC(np, int,
5847          ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5848     ALLOC(ixs, int,
5849          atm->np);
5850     ALLOC(iys, int,
5851          atm->np);
5852     ALLOC(izs, int,
5853          atm->np);
5854
5855     /* Set grid box size... */
5856     double dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
5857     double dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
5858     double dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
5859
5860     /* Set vertical coordinates... */
5861     #pragma omp parallel for default(shared)
5862     for (int iz = 0; iz < ctl->grid_nz; iz++) {
5863         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
5864         press[iz] = P(z[iz]);
5865     }
5866
5867     /* Set horizontal coordinates... */
5868     for (int ix = 0; ix < ctl->grid_nx; ix++)
5869         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
5870     #pragma omp parallel for default(shared)
5871     for (int iy = 0; iy < ctl->grid_ny; iy++) {
5872         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
5873         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
5874             * cos(lat[iy] * M_PI / 180.);
5875     }
5876
5877     /* Set time interval for output... */
5878     double t0 = t - 0.5 * ctl->dt_mod;
5879     double t1 = t + 0.5 * ctl->dt_mod;
5880
5881     /* Get grid box indices... */
5882     #pragma omp parallel for default(shared)
5883     for (int ip = 0; ip < atm->np; ip++) {
5884         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
5885         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);

```

```

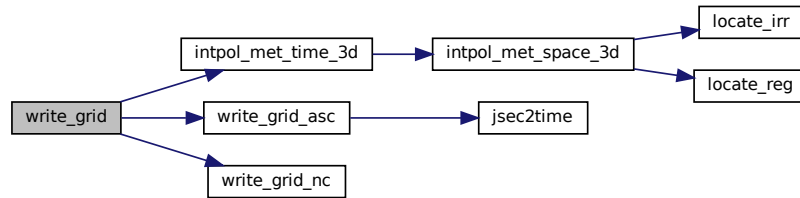
05886     izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05887     if (atm->time[ip] < t0 || atm->time[ip] > t1
05888         || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05889         || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05890         || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05891         izs[ip] = -1;
05892 }
05893
05894 /* Average data... */
05895 for (int ip = 0; ip < atm->np; ip++)
05896     if (izs[ip] >= 0) {
05897         int idx =
05898             ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
05899         np[idx]++;
05900         if (ctl->qnt_m >= 0)
05901             mass[idx] += atm->q[ctl->qnt_m][ip];
05902         if (ctl->qnt_vmr >= 0)
05903             vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
05904     }
05905
05906 /* Calculate column density and vmr... */
05907 #pragma omp parallel for default(shared)
05908 for (int ix = 0; ix < ctl->grid_nx; ix++)
05909     for (int iy = 0; iy < ctl->grid_ny; iy++)
05910         for (int iz = 0; iz < ctl->grid_nz; iz++) {
05911
05912             /* Get grid index... */
05913             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
05914
05915             /* Calculate column density... */
05916             cd[idx] = GSL_NAN;
05917             if (ctl->qnt_m >= 0)
05918                 cd[idx] = mass[idx] / (1e6 * area[iy]);
05919
05920             /* Calculate volume mixing ratio (implicit)... */
05921             vmr_impl[idx] = GSL_NAN;
05922             if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05923                 vmr_impl[idx] = 0;
05924                 if (mass[idx] > 0) {
05925
05926                     /* Get temperature... */
05927                     double temp;
05928                     INTPOL_INIT;
05929                     intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05930                                     lon[ix], lat[iy], &temp, ci, cw, 1);
05931
05932                     /* Calculate volume mixing ratio... */
05933                     vmr_impl[idx] = MA / ctl->molmass * mass[idx]
05934                                     / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
05935                 }
05936             }
05937
05938             /* Calculate volume mixing ratio (explicit)... */
05939             if (ctl->qnt_vmr >= 0 && np[idx] > 0)
05940                 vmr_expl[idx] /= np[idx];
05941             else
05942                 vmr_expl[idx] = GSL_NAN;
05943         }
05944
05945 /* Write ASCII data... */
05946 if (ctl->grid_type == 0)
05947     write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
05948                   t, z, lon, lat, area, dz, np);
05949
05950 /* Write netCDF data... */
05951 else if (ctl->grid_type == 1)
05952     write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
05953                  t, z, lon, lat, area, dz, np);
05954
05955 /* Error message... */
05956 else
05957     ERRMSG("Grid data format GRID_TYPE unknown!");
05958
05959 /* Free... */
05960 free(cd);
05961 free(mass);
05962 free(vmr_expl);
05963 free(vmr_impl);
05964 free(z);
05965 free(lon);
05966 free(lat);
05967 free(area);
05968 free(press);
05969 free(np);
05970 free(ixs);
05971 free(iys);
05972 free(izs);

```



```
05973 }
```

Here is the call graph for this function:



5.21.2.74 write_grid_asc() void write_grid_asc (
 const char * filename,
 ctl_t * ctl,
 double * cd,
 double * vmr_expl,
 double * vmr_impl,
 double t,
 double * z,
 double * lon,
 double * lat,
 double * area,
 double dz,
 int * np)

Write gridded data in ASCII format.

Definition at line 5977 of file libtrac.c.

```

05989     {
05990
05991     FILE *in, *out;
05992
05993     char line[LEN];
05994
05995     /* Check if gnuplot output is requested... */
05996     if (ctl->grid_gpfile[0] != '-') {
05997
05998         /* Create gnuplot pipe... */
05999         if (!(out = popen("gnuplot", "w")))
06000             ERRMSG("Cannot create pipe to gnuplot!");
06001
06002         /* Set plot filename... */
06003         fprintf(out, "set out \"%s.png\"\n", filename);
06004
06005         /* Set time string... */
06006         double r;
06007         int year, mon, day, hour, min, sec;
06008         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
06009         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
06010             year, mon, day, hour, min);
06011
06012         /* Dump gnuplot file to pipe... */
06013         if (!(in = fopen(ctl->grid_gpfile, "r")))
06014             ERRMSG("Cannot open file!");
06015         while (fgets(line, LEN, in))
06016             fprintf(out, "%s", line);
06017         fclose(in);
06018     }
06019
06020     else {
```

```

06021
06022     /* Create file... */
06023     if (!out = fopen(filename, "w"))
06024         ERRMSG("Cannot create file!");
06025 }
06026
06027 /* Write header... */
06028 fprintf(out,
06029     "# $1 = time [s]\n"
06030     "# $2 = altitude [km]\n"
06031     "# $3 = longitude [deg]\n"
06032     "# $4 = latitude [deg]\n"
06033     "# $5 = surface area [km^2]\n"
06034     "# $6 = layer depth [km]\n"
06035     "# $7 = number of particles [l]\n"
06036     "# $8 = column density (implicit) [kg/m^2]\n"
06037     "# $9 = volume mixing ratio (implicit) [ppv]\n"
06038     "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
06039
06040 /* Write data... */
06041 for (int ix = 0; ix < ctl->grid_nx; ix++) {
06042     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
06043         fprintf(out, "\n");
06044     for (int iy = 0; iy < ctl->grid_ny; iy++) {
06045         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
06046             fprintf(out, "\n");
06047         for (int iz = 0; iz < ctl->grid_nz; iz++) {
06048             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
06049             if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
06050                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
06051                     t, z[iz], lon[ix], lat[iy], area[iy], dz,
06052                     np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
06053         }
06054     }
06055 }
06056
06057 /* Close file... */
06058 fclose(out);
06059 }

```

Here is the call graph for this function:



5.21.2.75 write_grid_nc() void write_grid_nc (

```

    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )

```

Write gridded data in netCDF format.

Definition at line 6063 of file libtrac.c.

```

60675     {
60676
60677     double *help;
60678
60679     int *help2, ncid, dimid[10], varid;
60680
60681     size_t start[2], count[2];
60682
60683     /* Allocate... */
60684     ALLOC(help, double,
60685           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
60686     ALLOC(help2, int,
60687           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
60688
60689     /* Create file... */
60690     nc_create(filename, NC_CLOBBER, &ncid);
60691
60692     /* Define dimensions... */
60693     NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
60694     NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[1]));
60695     NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
60696     NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[3]));
60697     NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));
60698
60699     /* Define variables and their attributes... */
60700     NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
60701              "seconds since 2000-01-01 00:00:00 UTC");
60702     NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[1], "altitude", "km");
60703     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
60704     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[3], "longitude", "degrees_east");
60705     NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[4], "layer depth", "km");
60706     NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
60707     NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
60708     NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
60709              "volume mixing ratio (implicit)", "ppv");
60710     NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
60711              "volume mixing ratio (explicit)", "ppv");
60712     NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");
60713
60714     /* End definitions... */
60715     NC(nc_enddef(ncid));
60716
60717     /* Write data... */
60718     NC_PUT_DOUBLE("time", &t, 0);
60719     NC_PUT_DOUBLE("lon", lon, 0);
60720     NC_PUT_DOUBLE("lat", lat, 0);
60721     NC_PUT_DOUBLE("z", z, 0);
60722     NC_PUT_DOUBLE("area", area, 0);
60723     NC_PUT_DOUBLE("dz", &dz, 0);
60724
60725     for (int ix = 0; ix < ctl->grid_nx; ix++)
60726         for (int iy = 0; iy < ctl->grid_ny; iy++)
60727             for (int iz = 0; iz < ctl->grid_nz; iz++)
60728                 help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60729                     cd[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60730     NC_PUT_DOUBLE("cd", help, 0);
60731
60732     for (int ix = 0; ix < ctl->grid_nx; ix++)
60733         for (int iy = 0; iy < ctl->grid_ny; iy++)
60734             for (int iz = 0; iz < ctl->grid_nz; iz++)
60735                 help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60736                     vmr_impl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60737     NC_PUT_DOUBLE("vmr_impl", help, 0);
60738
60739     for (int ix = 0; ix < ctl->grid_nx; ix++)
60740         for (int iy = 0; iy < ctl->grid_ny; iy++)
60741             for (int iz = 0; iz < ctl->grid_nz; iz++)
60742                 help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60743                     vmr_expl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60744     NC_PUT_DOUBLE("vmr_expl", help, 0);
60745
60746     for (int ix = 0; ix < ctl->grid_nx; ix++)
60747         for (int iy = 0; iy < ctl->grid_ny; iy++)
60748             for (int iz = 0; iz < ctl->grid_nz; iz++)
60749                 help2[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60750                     np[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60751     NC_PUT_INT("np", help2, 0);
60752
60753     /* Close file... */
60754     NC(nc_close(ncid));
60755
60756     /* Free... */
60757     free(help);
60758     free(help2);
60759 }

```

```

5.21.2.76 write_met() int write_met (
    char * filename,
    ctl_t * ctl,
    met_t * met )

```

Read meteo data file.

Definition at line 6163 of file libtrac.c.

```

6166         {
6167
6168         /* Set timer... */
6169         SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);
6170
6171         /* Write info... */
6172         LOG(1, "Write meteo data: %s", filename);
6173
6174         /* Check compression flags... */
6175 #ifndef ZFP
6176         if (ctl->met_type == 3)
6177             ERRMSG("zfp compression not supported!");
6178 #endif
6179 #ifndef ZSTD
6180         if (ctl->met_type == 4)
6181             ERRMSG("zstd compression not supported!");
6182 #endif
6183
6184         /* Write binary... */
6185         if (ctl->met_type >= 1 && ctl->met_type <= 4) {
6186
6187             /* Create file... */
6188             FILE *out;
6189             if (!(out = fopen(filename, "w")))
6190                 ERRMSG("Cannot create file!");
6191
6192             /* Write type of binary data... */
6193             FWRITE(&ctl->met_type, int,
6194                 1,
6195                 out);
6196
6197             /* Write version of binary data... */
6198             int version = 100;
6199             FWRITE(&version, int,
6200                 1,
6201                 out);
6202
6203             /* Write grid data... */
6204             FWRITE(&met->time, double,
6205                 1,
6206                 out);
6207             FWRITE(&met->nx, int,
6208                 1,
6209                 out);
6210             FWRITE(&met->ny, int,
6211                 1,
6212                 out);
6213             FWRITE(&met->np, int,
6214                 1,
6215                 out);
6216             FWRITE(met->lon, double,
6217                 (size_t) met->nx,
6218                 out);
6219             FWRITE(met->lat, double,
6220                 (size_t) met->ny,
6221                 out);
6222             FWRITE(met->p, double,
6223                 (size_t) met->np,
6224                 out);
6225
6226             /* Write surface data... */
6227             write_met_bin_2d(out, met, met->ps, "PS");
6228             write_met_bin_2d(out, met, met->ts, "TS");
6229             write_met_bin_2d(out, met, met->zs, "ZS");
6230             write_met_bin_2d(out, met, met->us, "US");
6231             write_met_bin_2d(out, met, met->vs, "VS");
6232             write_met_bin_2d(out, met, met->pbl, "PBL");
6233             write_met_bin_2d(out, met, met->pt, "PT");
6234             write_met_bin_2d(out, met, met->tt, "TT");
6235             write_met_bin_2d(out, met, met->zt, "ZT");
6236             write_met_bin_2d(out, met, met->h2ot, "H2OT");

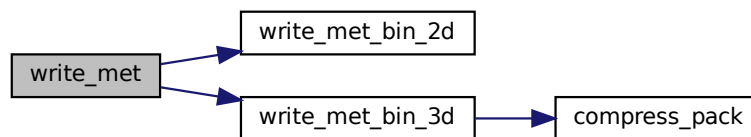
```

```

06237     write_met_bin_2d(out, met, met->pct, "PCT");
06238     write_met_bin_2d(out, met, met->pcb, "PCB");
06239     write_met_bin_2d(out, met, met->cl, "CL");
06240     write_met_bin_2d(out, met, met->plcl, "PLCL");
06241     write_met_bin_2d(out, met, met->plfc, "PLFC");
06242     write_met_bin_2d(out, met, met->pel, "PEL");
06243     write_met_bin_2d(out, met, met->cape, "CAPE");
06244     write_met_bin_2d(out, met, met->cin, "CIN");
06245
06246     /* Write level data... */
06247     write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
06248     write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
06249     write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
06250     write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
06251     write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
06252     write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
06253     write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
06254     write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
06255     write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
06256     write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);
06257
06258     /* Write final flag... */
06259     int final = 999;
06260     FWRITE(&final, int,
06261           1,
06262           out);
06263
06264     /* Close file... */
06265     fclose(out);
06266 }
06267
06268 return 0;
06269 }

```

Here is the call graph for this function:



5.21.2.77 write_met_bin_2d() void write_met_bin_2d (

```

FILE * out,
met_t * met,
float var[EX][EY],
char * varname )

```

Write 2-D meteo variable.

Definition at line 6273 of file libtrac.c.

```

06277     {
06278
06279     float *help;
06280
06281     /* Allocate... */
06282     ALLOC(help, float,
06283           EX * EY);
06284
06285     /* Copy data... */
06286     for (int ix = 0; ix < met->nx; ix++)
06287         for (int iy = 0; iy < met->ny; iy++)
06288             help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];

```

```

06289
06290 /* Write uncompressed data... */
06291 LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
06292 FWRITE(help, float,
06293         (size_t) (met->nx * met->ny),
06294         out);
06295
06296 /* Free... */
06297 free(help);
06298 }

```

5.21.2.78 write_met_bin_3d() void write_met_bin_3d (

```

FILE * out,
ctl_t * ctl,
met_t * met,
float var[EX][EY][EP],
char * varname,
int precision,
double tolerance )

```

Write 3-D meteo variable.

Definition at line 6302 of file libtrac.c.

```

06309 {
06310
06311     float *help;
06312
06313     /* Allocate... */
06314     ALLOC(help, float,
06315           EX * EY * EP);
06316
06317     /* Copy data... */
06318 #pragma omp parallel for default(shared) collapse(2)
06319     for (int ix = 0; ix < met->nx; ix++)
06320         for (int iy = 0; iy < met->ny; iy++)
06321             for (int ip = 0; ip < met->np; ip++)
06322                 help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];
06323
06324     /* Write uncompressed data... */
06325     if (ctl->met_type == 1) {
06326         LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
06327         FWRITE(help, float,
06328               (size_t) (met->nx * met->ny * met->np),
06329               out);
06330     }
06331
06332     /* Write packed data... */
06333     else if (ctl->met_type == 2)
06334         compress_pack(varname, help, (size_t) (met->ny * met->nx),
06335                      (size_t) met->np, 0, out);
06336
06337     /* Write zfp data... */
06338 #ifdef ZFP
06339     else if (ctl->met_type == 3)
06340         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
06341                     tolerance, 0, out);
06342 #endif
06343
06344     /* Write zstd data... */
06345 #ifdef ZSTD
06346     else if (ctl->met_type == 4)
06347         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
06348                      out);
06349 #endif
06350
06351     /* Unknown method... */
06352     else {
06353         ERRMSG("MET_TYPE not supported!");
06354         LOG(3, "%d %g", precision, tolerance);
06355     }
06356
06357     /* Free... */
06358     free(help);
06359 }

```

Here is the call graph for this function:



5.21.2.79 write_prof() void write_prof (

 const char * filename,

 ctl_t * ctl,

 met_t * met0,

 met_t * met1,

 atm_t * atm,

 double t)

Write profile data.

Definition at line 6363 of file libtrac.c.

```

06369     {
06370
06371     static FILE *out;
06372
06373     static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
06374         dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;
06375
06376     static int nob, *obscount, ip, okay;
06377
06378     /* Set timer... */
06379     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
06380
06381     /* Init... */
06382     if (t == ctl->t_start) {
06383
06384         /* Check quantity index for mass... */
06385         if (ctl->qnt_m < 0)
06386             ERRMSG("Need quantity mass!");
06387
06388         /* Check molar mass... */
06389         if (ctl->molmass <= 0)
06390             ERRMSG("Specify molar mass!");
06391
06392         /* Allocate... */
06393         ALLOC(lon, double,
06394             ctl->prof_nx);
06395         ALLOC(lat, double,
06396             ctl->prof_ny);
06397         ALLOC(area, double,
06398             ctl->prof_ny);
06399         ALLOC(z, double,
06400             ctl->prof_nz);
06401         ALLOC(press, double,
06402             ctl->prof_nz);
06403         ALLOC(rt, double,
06404             NOBS);
06405         ALLOC(rz, double,
06406             NOBS);
06407         ALLOC(rlon, double,
06408             NOBS);
06409         ALLOC(rlat, double,
06410             NOBS);
06411         ALLOC(robs, double,
06412             NOBS);
06413
06414         /* Read observation data... */
  
```

```

06415     read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06416
06417     /* Create new output file... */
06418     LOG(1, "Write profile data: %s", filename);
06419     if (!out = fopen(filename, "w"))
06420         ERRMSG("Cannot create file!");
06421
06422     /* Write header... */
06423     fprintf(out,
06424             "# $1 = time [s]\n"
06425             "# $2 = altitude [km]\n"
06426             "# $3 = longitude [deg]\n"
06427             "# $4 = latitude [deg]\n"
06428             "# $5 = pressure [hPa]\n"
06429             "# $6 = temperature [K]\n"
06430             "# $7 = volume mixing ratio [ppv]\n"
06431             "# $8 = H2O volume mixing ratio [ppv]\n"
06432             "# $9 = O3 volume mixing ratio [ppv]\n"
06433             "# $10 = observed BT index [K]\n"
06434             "# $11 = number of observations\n");
06435
06436     /* Set grid box size... */
06437     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
06438     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
06439     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
06440
06441     /* Set vertical coordinates... */
06442     for (int iz = 0; iz < ctl->prof_nz; iz++) {
06443         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
06444         press[iz] = P(z[iz]);
06445     }
06446
06447     /* Set horizontal coordinates... */
06448     for (int ix = 0; ix < ctl->prof_nx; ix++)
06449         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
06450     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06451         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
06452         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
06453             * cos(lat[iy] * M_PI / 180.);
06454     }
06455 }
06456
06457 /* Set time interval... */
06458 double t0 = t - 0.5 * ctl->dt_mod;
06459 double t1 = t + 0.5 * ctl->dt_mod;
06460
06461 /* Allocate... */
06462 ALLOC(mass, double,
06463       ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
06464 ALLOC(obsmean, double,
06465       ctl->prof_nx * ctl->prof_ny);
06466 ALLOC(obscount, int,
06467       ctl->prof_nx * ctl->prof_ny);
06468
06469 /* Loop over observations... */
06470 for (int i = 0; i < nobs; i++) {
06471
06472     /* Check time... */
06473     if (rt[i] < t0)
06474         continue;
06475     else if (rt[i] >= t1)
06476         break;
06477
06478     /* Check observation data... */
06479     if (!isfinite(robs[i]))
06480         continue;
06481
06482     /* Calculate indices... */
06483     int ix = (int) ((rlon[i] - ctl->prof_lon0) / dlon);
06484     int iy = (int) ((rlat[i] - ctl->prof_lat0) / dlat);
06485
06486     /* Check indices... */
06487     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
06488         continue;
06489
06490     /* Get mean observation index... */
06491     int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
06492     obsmean[idx] += robs[i];
06493     obscount[idx]++;
06494 }
06495
06496 /* Analyze model data... */
06497 for (ip = 0; ip < atm->np; ip++) {
06498
06499     /* Check time... */
06500     if (atm->time[ip] < t0 || atm->time[ip] > t1)
06501         continue;

```

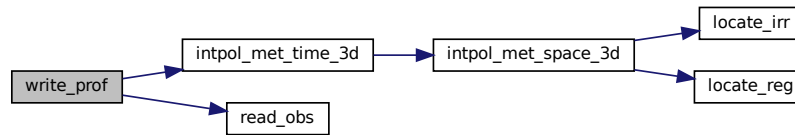


```

06502
06503 /* Get indices... */
06504 int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
06505 int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
06506 int iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
06507
06508 /* Check indices... */
06509 if (ix < 0 || ix >= ctl->prof_nx ||
06510     iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
06511     continue;
06512
06513 /* Get total mass in grid cell... */
06514 int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06515 mass[idx] += atm->q[ctl->qnt_m][ip];
06516 }
06517
06518 /* Extract profiles... */
06519 for (int ix = 0; ix < ctl->prof_nx; ix++)
06520     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06521         int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
06522         if (obscount[idx2] > 0) {
06523
06524             /* Check profile... */
06525             okay = 0;
06526             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06527                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06528                 if (mass[idx3] > 0) {
06529                     okay = 1;
06530                     break;
06531                 }
06532             }
06533             if (!okay)
06534                 continue;
06535
06536             /* Write output... */
06537             fprintf(out, "\n");
06538
06539             /* Loop over altitudes... */
06540             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06541
06542                 /* Get temperature, water vapor, and ozone... */
06543                 INTPOL_INIT;
06544                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
06545                                     lon[ix], lat[iy], &temp, ci, cw, 1);
06546                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
06547                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
06548                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
06549                                     lon[ix], lat[iy], &o3, ci, cw, 0);
06550
06551                 /* Calculate volume mixing ratio... */
06552                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06553                 vmr = MA / ctl->molmass * mass[idx3]
06554                     / (RHO(press[iz], temp) * area[iy] * dz * 1e9);
06555
06556                 /* Write output... */
06557                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
06558                         t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
06559                         obsmean[idx2] / obscount[idx2], obscount[idx2]);
06560             }
06561         }
06562     }
06563
06564 /* Free... */
06565 free(mass);
06566 free(obsmean);
06567 free(obscount);
06568
06569 /* Finalize... */
06570 if (t == ctl->t_stop) {
06571
06572     /* Close output file... */
06573     fclose(out);
06574
06575     /* Free... */
06576     free(lon);
06577     free(lat);
06578     free(area);
06579     free(z);
06580     free(press);
06581     free(rt);
06582     free(rz);
06583     free(rlon);
06584     free(rlat);
06585     free(robs);
06586 }
06587 }

```

Here is the call graph for this function:



5.21.2.80 write_sample() void write_sample (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write sample data.

Definition at line 6591 of file libtrac.c.

```

06597     {
06598
06599     static FILE *out;
06600
06601     static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;
06602
06603     static int nobs;
06604
06605     /* Set timer... */
06606     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
06607
06608     /* Init... */
06609     if (t == ctl->t_start) {
06610
06611         /* Allocate... */
06612         ALLOC(rt, double,
06613             NOBS);
06614         ALLOC(rz, double,
06615             NOBS);
06616         ALLOC(rlon, double,
06617             NOBS);
06618         ALLOC(rlat, double,
06619             NOBS);
06620         ALLOC(robs, double,
06621             NOBS);
06622
06623         /* Read observation data... */
06624         read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06625
06626         /* Create output file... */
06627         LOG(1, "Write sample data: %s", filename);
06628         if (!(out = fopen(filename, "w")))
06629             ERRMSG("Cannot create file!");
06630
06631         /* Write header... */
06632         fprintf(out,
06633             "# $1 = time [s]\n"
06634             "# $2 = altitude [km]\n"
06635             "# $3 = longitude [deg]\n"
06636             "# $4 = latitude [deg]\n"
06637             "# $5 = surface area [km^2]\n"
06638             "# $6 = layer depth [km]\n"
06639             "# $7 = number of particles [l]\n"
06640             "# $8 = column density [kg/m^2]\n"
06641             "# $9 = volume mixing ratio [ppv]\n"
06642             "# $10 = observed BT index [K]\n\n");
06643

```

```

06644     /* Set latitude range, squared radius, and area... */
06645     dlat = DY2DEG(ctl->sample_dx);
06646     rmax2 = SQR(ctl->sample_dx);
06647     area = M_PI * rmax2;
06648 }
06649
06650 /* Set time interval for output... */
06651 double t0 = t - 0.5 * ctl->dt_mod;
06652 double t1 = t + 0.5 * ctl->dt_mod;
06653
06654 /* Loop over observations... */
06655 for (int i = 0; i < nob; i++) {
06656
06657     /* Check time... */
06658     if (rt[i] < t0)
06659         continue;
06660     else if (rt[i] >= t1)
06661         break;
06662
06663     /* Calculate Cartesian coordinates... */
06664     double x0[3];
06665     geo2cart(0, rlon[i], rlat[i], x0);
06666
06667     /* Set pressure range... */
06668     double rp = P(rz[i]);
06669     double ptop = P(rz[i] + ctl->sample_dz);
06670     double pbot = P(rz[i] - ctl->sample_dz);
06671
06672     /* Init... */
06673     double mass = 0;
06674     int np = 0;
06675
06676     /* Loop over air parcels... */
06677 #pragma omp parallel for default(shared) reduction(+:mass,np)
06678     for (int ip = 0; ip < atm->np; ip++) {
06679
06680         /* Check time... */
06681         if (atm->time[ip] < t0 || atm->time[ip] > t1)
06682             continue;
06683
06684         /* Check latitude... */
06685         if (fabs(rlat[i] - atm->lat[ip]) > dlat)
06686             continue;
06687
06688         /* Check horizontal distance... */
06689         double x1[3];
06690         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06691         if (DIST2(x0, x1) > rmax2)
06692             continue;
06693
06694         /* Check pressure... */
06695         if (ctl->sample_dz > 0)
06696             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
06697                 continue;
06698
06699         /* Add mass... */
07000         if (ctl->qnt_m >= 0)
07001             mass += atm->q[ctl->qnt_m][ip];
07002         np++;
07003     }
07004
07005     /* Calculate column density... */
07006     double cd = mass / (1e6 * area);
07007
07008     /* Calculate volume mixing ratio... */
07009     double vmr = 0;
07010     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
07011         if (mass > 0) {
07012
07013             /* Get temperature... */
07014             double temp;
07015             INTPOL_INIT;
07016             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
07017                               rlon[i], rlat[i], &temp, ci, cw, 1);
07018
07019             /* Calculate volume mixing ratio... */
07020             vmr = MA / ctl->molmass * mass
07021                  / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
07022         }
07023     } else
07024         vmr = GSL_NAN;
07025
07026     /* Write output... */
07027     fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
07028             rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
07029 }
07030

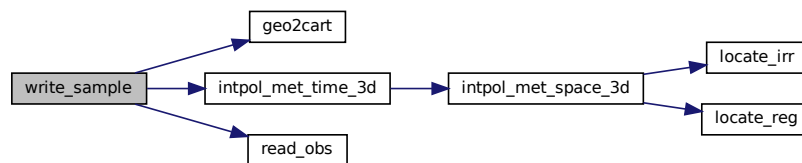
```

```

06731  /* Finalize..... */
06732  if (t == ctl->t_stop) {
06733
06734      /* Close output file... */
06735      fclose(out);
06736
06737      /* Free... */
06738      free(rt);
06739      free(rz);
06740      free(rlon);
06741      free(rlat);
06742      free(robs);
06743  }
06744 }

```

Here is the call graph for this function:



5.21.2.81 write_station() void write_station (

```

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

```

Write station data.

Definition at line 6748 of file libtrac.c.

```

06752  {
06753
06754      static FILE *out;
06755
06756      static double rmax2, x0[3], x1[3];
06757
06758      /* Set timer... */
06759      SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
06760
06761      /* Init... */
06762      if (t == ctl->t_start) {
06763
06764          /* Write info... */
06765          LOG(1, "Write station data: %s", filename);
06766
06767          /* Create new file... */
06768          if (!(out = fopen(filename, "w")))
06769              ERRMSG("Cannot create file!");
06770
06771          /* Write header... */
06772          fprintf(out,
06773              "# $1 = time [s]\n"
06774              "# $2 = altitude [km]\n"
06775              "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
06776          for (int iq = 0; iq < ctl->nq; iq++)
06777              fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
06778                  ctl->qnt_name[iq], ctl->qnt_unit[iq]);
06779          fprintf(out, "\n");
06780
06781          /* Set geolocation and search radius... */
06782          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
06783          rmax2 = SQR(ctl->stat_r);
06784      }

```

```

06785
06786  /* Set time interval for output... */
06787  double t0 = t - 0.5 * ctl->dt_mod;
06788  double t1 = t + 0.5 * ctl->dt_mod;
06789
06790  /* Loop over air parcels... */
06791  for (int ip = 0; ip < atm->np; ip++) {
06792
06793      /* Check time... */
06794      if (atm->time[ip] < t0 || atm->time[ip] > t1)
06795          continue;
06796
06797      /* Check time range for station output... */
06798      if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
06799          continue;
06800
06801      /* Check station flag... */
06802      if (ctl->qnt_stat >= 0)
06803          if (atm->q[ctl->qnt_stat][ip])
06804              continue;
06805
06806      /* Get Cartesian coordinates... */
06807      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06808
06809      /* Check horizontal distance... */
06810      if (DIST2(x0, x1) > rmax2)
06811          continue;
06812
06813      /* Set station flag... */
06814      if (ctl->qnt_stat >= 0)
06815          atm->q[ctl->qnt_stat][ip] = 1;
06816
06817      /* Write data... */
06818      fprintf(out, "%.2f %g %g %g",
06819              atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
06820      for (int iq = 0; iq < ctl->nq; iq++) {
06821          fprintf(out, " ");
06822          fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
06823      }
06824      fprintf(out, "\n");
06825  }
06826
06827  /* Close file... */
06828  if (t == ctl->t_stop)
06829      fclose(out);
06830  }

```

Here is the call graph for this function:



5.22 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) 2013-2023 Forschungszentrum Juelich GmbH

```

```

00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*****
00028
00029 double buoyancy_frequency(
00030     double p0,
00031     double t0,
00032     double p1,
00033     double t1) {
00034
00035     double theta0 = THETA(p0, t0);
00036     double theta1 = THETA(p1, t1);
00037     double dz = RI / MA / GO * 0.5 * (t0 + t1) * (log(p0) - log(p1));
00038
00039     return sqrt(2. * GO / (theta0 + theta1) * (theta1 - theta0) / dz);
00040 }
00041
00042 /*****
00043
00044 void cart2geo(
00045     double *x,
00046     double *z,
00047     double *lon,
00048     double *lat) {
00049
00050     double radius = NORM(x);
00051     *lat = asin(x[2] / radius) * 180. / M_PI;
00052     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00053     *z = radius - RE;
00054 }
00055
00056 /*****
00057
00058 double clim_hno3(
00059     clim_t * clim,
00060     double t,
00061     double lat,
00062     double p) {
00063
00064     /* Get seconds since begin of year... */
00065     double sec = FMOD(t, 365.25 * 86400.);
00066     while (sec < 0)
00067         sec += 365.25 * 86400.;
00068
00069     /* Check pressure... */
00070     if (p < clim->hno3_p[0])
00071         p = clim->hno3_p[0];
00072     else if (p > clim->hno3_p[clim->hno3_np - 1])
00073         p = clim->hno3_p[clim->hno3_np - 1];
00074
00075     /* Check latitude... */
00076     if (lat < clim->hno3_lat[0])
00077         lat = clim->hno3_lat[0];
00078     else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
00079         lat = clim->hno3_lat[clim->hno3_nlat - 1];
00080
00081     /* Get indices... */
00082     int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
00083     int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
00084     int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);
00085
00086     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00087     double aux00 = LIN(clim->hno3_p[ip],
00088         clim->hno3[isec][ilat][ip],
00089         clim->hno3_p[ip + 1],
00090         clim->hno3[isec][ilat][ip + 1], p);
00091     double aux01 = LIN(clim->hno3_p[ip],
00092         clim->hno3[isec][ilat + 1][ip],
00093         clim->hno3_p[ip + 1],
00094         clim->hno3[isec][ilat + 1][ip + 1], p);
00095     double aux10 = LIN(clim->hno3_p[ip],
00096         clim->hno3[isec + 1][ilat][ip],
00097         clim->hno3_p[ip + 1],
00098         clim->hno3[isec + 1][ilat][ip + 1], p);
00099     double aux11 = LIN(clim->hno3_p[ip],
00100         clim->hno3[isec + 1][ilat + 1][ip],
00101         clim->hno3_p[ip + 1],
00102         clim->hno3[isec + 1][ilat + 1][ip + 1], p);
00103     aux00 = LIN(clim->hno3_lat[ilat], aux00,
00104         clim->hno3_lat[ilat + 1], aux01, lat);
00105     aux11 = LIN(clim->hno3_lat[ilat], aux10,
00106         clim->hno3_lat[ilat + 1], aux11, lat);
00107     aux00 = LIN(clim->hno3_time[isec], aux00,
00108         clim->hno3_time[isec + 1], aux11, sec);
00109

```

```

00110  /* Convert from ppb to ppv... */
00111  return GSL_MAX(1e-9 * aux00, 0.0);
00112 }
00113
00114 /*****
00115
00116 void clim_hno3_init(
00117     clim_t * clim) {
00118
00119     /* Write info... */
00120     LOG(1, "Initialize HNO3 data...");
00121
00122     clim->hno3_ntime = 12;
00123     double hno3_time[12] = {
00124         1209600.00, 3888000.00, 6393600.00,
00125         9072000.00, 11664000.00, 14342400.00,
00126         16934400.00, 19612800.00, 22291200.00,
00127         24883200.00, 27561600.00, 30153600.00
00128     };
00129     memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));
00130
00131     clim->hno3_nlat = 18;
00132     double hno3_lat[18] = {
00133         -85, -75, -65, -55, -45, -35, -25, -15, -5,
00134         5, 15, 25, 35, 45, 55, 65, 75, 85
00135     };
00136     memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));
00137
00138     clim->hno3_np = 10;
00139     double hno3_p[10] = {
00140         4.64159, 6.81292, 10, 14.678, 21.5443,
00141         31.6228, 46.4159, 68.1292, 100, 146.78
00142     };
00143     memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));
00144
00145     double hno3[12][18][10] = {
00146         {0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00147         {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00148         {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00149         {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00150         {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00151         {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00152         {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00153         {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00154         {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00155         {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00156         {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00157         {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00158         {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00159         {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00160         {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00161         {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00162         {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00163         {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19},
00164         {0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00165         {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00166         {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00167         {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00168         {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00169         {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00170         {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00171         {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00172         {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00173         {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00174         {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00175         {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00176         {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00177         {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00178         {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00179         {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00180         {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00181         {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17},
00182         {1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00183         {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00184         {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00185         {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00186         {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00187         {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00188         {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00189         {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00190         {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00191         {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00192         {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00193         {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00194         {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00195         {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00196         {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},

```

```
00197 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00198 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00199 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00200 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75}},
00201 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00202 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00203 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00204 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00205 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00206 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00207 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00208 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00209 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00210 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00211 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00212 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00213 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00214 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00215 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00216 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00217 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00218 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63}},
00219 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00220 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00221 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00222 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00223 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00224 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00225 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00226 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00227 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00228 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00229 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00230 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00231 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00232 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00233 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00234 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00235 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00236 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26}},
00237 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00238 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00239 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00240 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00241 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00242 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00243 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00244 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
00245 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
00246 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
00247 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
00248 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
00249 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
00250 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
00251 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
00252 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
00253 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00254 {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}},
00255 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
00256 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
00257 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
00258 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
00259 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
00260 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
00261 {1.02, 2.3, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
00262 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
00263 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
00264 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
00265 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
00266 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
00267 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
00268 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
00269 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
00270 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
00271 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00272 {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}},
00273 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
00274 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
00275 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
00276 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
00277 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
00278 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
00279 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
00280 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
00281 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
00282 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
00283 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
```



```

00284     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00285     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00286     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00287     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00288     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00289     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00290     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00291     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
00292     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00293     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00294     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00295     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00296     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00297     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00298     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00299     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00300     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00301     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00302     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00303     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00304     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00305     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00306     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00307     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00308     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00309     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
00310     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00311     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00312     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00313     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00314     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00315     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00316     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00317     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00318     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00319     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00320     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00321     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00322     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00323     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00324     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00325     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8},
00326     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00327     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00328     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00329     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00330     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00331     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00332     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00333     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00334     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00335     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00336     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00337     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00338     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00339     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00340     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00341     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00342     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00343     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05},
00344     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00345     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00346     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00347     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00348     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00349     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00350     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00351     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00352     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00353     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00354     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00355     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00356     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00357     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00358     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00359     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00360     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00361     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00362     };
00363     memcpy(clim->hno3, hno3, sizeof(clim->hno3));
00364
00365     /* Get range... */
00366     double hno3min = 1e99, hno3max = -1e99;
00367     for (int it = 0; it < clim->hno3_ntime; it++)
00368         for (int iz = 0; iz < clim->hno3_npz; iz++)
00369             for (int iy = 0; iy < clim->hno3_nlat; iy++) {
00370                 hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);

```

```

00371         hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
00372     }
00373
00374     /* Write info... */
00375     LOG(2, "Number of time steps: %d", clim->hno3_ntime);
00376     LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00377         clim->hno3_time[0], clim->hno3_time[1],
00378         clim->hno3_time[clim->hno3_ntime - 1]);
00379     LOG(2, "Number of pressure levels: %d", clim->hno3_np);
00380     LOG(2, "Altitude levels: %g, %g ... %g km",
00381         Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
00382         Z(clim->hno3_p[clim->hno3_np - 1]));
00383     LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
00384         clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
00385     LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
00386     LOG(2, "Latitudes: %g, %g ... %g deg",
00387         clim->hno3_lat[0], clim->hno3_lat[1],
00388         clim->hno3_lat[clim->hno3_nlat - 1]);
00389     LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
00390         1e-9 * hno3max);
00391 }
00392
00393 /*****
00394
00395 double clim_oh(
00396     clim_t * clim,
00397     double t,
00398     double lat,
00399     double p) {
00400
00401     /* Get seconds since begin of year... */
00402     double sec = FMOD(t, 365.25 * 86400.);
00403     while (sec < 0)
00404         sec += 365.25 * 86400.;
00405
00406     /* Check pressure... */
00407     if (p < clim->oh_p[clim->oh_np - 1])
00408         p = clim->oh_p[clim->oh_np - 1];
00409     else if (p > clim->oh_p[0])
00410         p = clim->oh_p[0];
00411
00412     /* Check latitude... */
00413     if (lat < clim->oh_lat[0])
00414         lat = clim->oh_lat[0];
00415     else if (lat > clim->oh_lat[clim->oh_nlat - 1])
00416         lat = clim->oh_lat[clim->oh_nlat - 1];
00417
00418     /* Get indices... */
00419     int isec = locate_irr(clim->oh_time, clim->oh_ntime, sec);
00420     int ilat = locate_reg(clim->oh_lat, clim->oh_nlat, lat);
00421     int ip = locate_irr(clim->oh_p, clim->oh_np, p);
00422
00423     /* Interpolate OH climatology... */
00424     double aux00 = LIN(clim->oh_p[ip],
00425         clim->oh[isec][ip][ilat],
00426         clim->oh_p[ip + 1],
00427         clim->oh[isec][ip + 1][ilat], p);
00428     double aux01 = LIN(clim->oh_p[ip],
00429         clim->oh[isec][ip][ilat + 1],
00430         clim->oh_p[ip + 1],
00431         clim->oh[isec][ip + 1][ilat + 1], p);
00432     double aux10 = LIN(clim->oh_p[ip],
00433         clim->oh[isec + 1][ip][ilat],
00434         clim->oh_p[ip + 1],
00435         clim->oh[isec + 1][ip + 1][ilat], p);
00436     double aux11 = LIN(clim->oh_p[ip],
00437         clim->oh[isec + 1][ip][ilat + 1],
00438         clim->oh_p[ip + 1],
00439         clim->oh[isec + 1][ip + 1][ilat + 1], p);
00440     aux00 = LIN(clim->oh_lat[ilat], aux00, clim->oh_lat[ilat + 1], aux01, lat);
00441     aux11 = LIN(clim->oh_lat[ilat], aux10, clim->oh_lat[ilat + 1], aux11, lat);
00442     aux00 =
00443         LIN(clim->oh_time[isec], aux00, clim->oh_time[isec + 1], aux11, sec);
00444
00445     return GSL_MAX(aux00, 0.0);
00446 }
00447
00448 /*****
00449
00450 double clim_oh_diurnal(
00451     ctl_t * ctl,
00452     clim_t * clim,
00453     double t,
00454     double p,
00455     double lon,
00456     double lat) {
00457

```

```

00458     double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);
00459
00460     if (sza2 <= M_PI / 2. * 89. / 90.)
00461         return oh * exp(-ctl->oh_chem_beta / cos(sza2));
00462     else
00463         return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
00464 }
00465
00466 /*****
00467 void clim_oh_init(
00468     ctl_t * ctl,
00469     clim_t * clim) {
00470
00471     int nt, ncid, varid;
00472
00473     double *help, ohmin = 1e99, ohmax = -1e99;
00474
00475     /* Write info... */
00476     LOG(1, "Read OH data: %s", ctl->clim_oh_filename);
00477
00478     /* Open netCDF file... */
00479     if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00480         WARN("OH climatology data are missing!");
00481         return;
00482     }
00483
00484     /* Read pressure data... */
00485     NC_INQ_DIM("press", &clim->oh_np, 2, CP);
00486     NC_GET_DOUBLE("press", clim->oh_p, 1);
00487
00488     /* Check ordering of pressure data... */
00489     if (clim->oh_p[0] < clim->oh_p[1])
00490         ERRMSG("Pressure data are not descending!");
00491
00492     /* Read latitudes... */
00493     NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
00494     NC_GET_DOUBLE("lat", clim->oh_lat, 1);
00495
00496     /* Check ordering of latitudes... */
00497     if (clim->oh_lat[0] > clim->oh_lat[1])
00498         ERRMSG("Latitude data are not ascending!");
00499
00500     /* Set time data for monthly means... */
00501     clim->oh_ntime = 12;
00502     clim->oh_time[0] = 1209600.00;
00503     clim->oh_time[1] = 3888000.00;
00504     clim->oh_time[2] = 6393600.00;
00505     clim->oh_time[3] = 9072000.00;
00506     clim->oh_time[4] = 11664000.00;
00507     clim->oh_time[5] = 14342400.00;
00508     clim->oh_time[6] = 16934400.00;
00509     clim->oh_time[7] = 19612800.00;
00510     clim->oh_time[8] = 22291200.00;
00511     clim->oh_time[9] = 24883200.00;
00512     clim->oh_time[10] = 27561600.00;
00513     clim->oh_time[11] = 30153600.00;
00514
00515     /* Check number of timesteps... */
00516     NC_INQ_DIM("time", &nt, 12, 12);
00517
00518     /* Read OH data... */
00519     ALLOC(help, double,
00520         clim->oh_nlat * clim->oh_np * clim->oh_ntime);
00521     NC_GET_DOUBLE("OH", help, 1);
00522     for (int it = 0; it < clim->oh_ntime; it++)
00523         for (int iz = 0; iz < clim->oh_np; iz++)
00524             for (int iy = 0; iy < clim->oh_nlat; iy++) {
00525                 clim->oh[it][iz][iy] =
00526                     help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
00527                     / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
00528                         clim->oh_lat[iy]);
00529                 ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
00530                 ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
00531             }
00532     free(help);
00533
00534     /* Close netCDF file... */
00535     NC(nc_close(ncid));
00536
00537     /* Write info... */
00538     LOG(2, "Number of time steps: %d", clim->oh_ntime);
00539     LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00540         clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
00541     LOG(2, "Number of pressure levels: %d", clim->oh_np);
00542     LOG(2, "Altitude levels: %g, %g ... %g km",
00543         Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));

```

```

00545     LOG(2, "Pressure levels: %g, %g ... %g hPa",
00546         clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
00547     LOG(2, "Number of latitudes: %d", clim->oh_nlat);
00548     LOG(2, "Latitudes: %g, %g ... %g deg",
00549         clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
00550     LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
00551 }
00552
00553 /*****
00554
00555 double clim_oh_init_help(
00556     double beta,
00557     double time,
00558     double lat) {
00559
00560     double aux, lon, sum = 0;
00561
00562     int n = 0;
00563
00564     /* Integrate day/night correction factor over longitude... */
00565     for (lon = -180; lon < 180; lon += 1) {
00566         aux = sza(time, lon, lat);
00567         if (aux <= M_PI / 2. * 85. / 90.)
00568             sum += exp(-beta / cos(aux));
00569         else
00570             sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
00571         n++;
00572     }
00573     return sum / (double) n;
00574 }
00575
00576 /*****
00577
00578 double clim_h2o2(
00579     clim_t * clim,
00580     double t,
00581     double lat,
00582     double p) {
00583
00584     /* Get seconds since begin of year... */
00585     double sec = FMOD(t, 365.25 * 86400.);
00586     while (sec < 0)
00587         sec += 365.25 * 86400.;
00588
00589     /* Check pressure... */
00590     if (p < clim->h2o2_p[clim->h2o2_np - 1])
00591         p = clim->h2o2_p[clim->h2o2_np - 1];
00592     else if (p > clim->h2o2_p[0])
00593         p = clim->h2o2_p[0];
00594
00595     /* Check latitude... */
00596     if (lat < clim->h2o2_lat[0])
00597         lat = clim->h2o2_lat[0];
00598     else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
00599         lat = clim->h2o2_lat[clim->h2o2_nlat - 1];
00600
00601     /* Get indices... */
00602     int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
00603     int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
00604     int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);
00605
00606     /* Interpolate H2O2 climatology... */
00607     double aux00 = LIN(clim->h2o2_p[ip],
00608         clim->h2o2[isec][ip][ilat],
00609         clim->h2o2_p[ip + 1],
00610         clim->h2o2[isec][ip + 1][ilat], p);
00611     double aux01 = LIN(clim->h2o2_p[ip],
00612         clim->h2o2[isec][ip][ilat + 1],
00613         clim->h2o2_p[ip + 1],
00614         clim->h2o2[isec][ip + 1][ilat + 1], p);
00615     double aux10 = LIN(clim->h2o2_p[ip],
00616         clim->h2o2[isec + 1][ip][ilat],
00617         clim->h2o2_p[ip + 1],
00618         clim->h2o2[isec + 1][ip + 1][ilat], p);
00619     double aux11 = LIN(clim->h2o2_p[ip],
00620         clim->h2o2[isec + 1][ip][ilat + 1],
00621         clim->h2o2_p[ip + 1],
00622         clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
00623     aux00 =
00624         LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
00625     aux11 =
00626         LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
00627     aux00 =
00628         LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);
00629
00630     return GSL_MAX(aux00, 0.0);
00631 }

```

```

00632
00633 /*****
00634
00635 void clim_h2o2_init(
00636     ctl_t * ctl,
00637     clim_t * clim) {
00638
00639     int ncid, varid, it, iy, iz, nt;
00640
00641     double *help, h2o2min = 1e99, h2o2max = -1e99;
00642
00643     /* Write info... */
00644     LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);
00645
00646     /* Open netCDF file... */
00647     if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00648         WARN("H2O2 climatology data are missing!");
00649         return;
00650     }
00651
00652     /* Read pressure data... */
00653     NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
00654     NC_GET_DOUBLE("press", clim->h2o2_p, 1);
00655
00656     /* Check ordering of pressure data... */
00657     if (clim->h2o2_p[0] < clim->h2o2_p[1])
00658         ERRMSG("Pressure data are not descending!");
00659
00660     /* Read latitudes... */
00661     NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
00662     NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);
00663
00664     /* Check ordering of latitude data... */
00665     if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
00666         ERRMSG("Latitude data are not ascending!");
00667
00668     /* Set time data (for monthly means)... */
00669     clim->h2o2_ntime = 12;
00670     clim->h2o2_time[0] = 1209600.00;
00671     clim->h2o2_time[1] = 3888000.00;
00672     clim->h2o2_time[2] = 6393600.00;
00673     clim->h2o2_time[3] = 9072000.00;
00674     clim->h2o2_time[4] = 11664000.00;
00675     clim->h2o2_time[5] = 14342400.00;
00676     clim->h2o2_time[6] = 16934400.00;
00677     clim->h2o2_time[7] = 19612800.00;
00678     clim->h2o2_time[8] = 22291200.00;
00679     clim->h2o2_time[9] = 24883200.00;
00680     clim->h2o2_time[10] = 27561600.00;
00681     clim->h2o2_time[11] = 30153600.00;
00682
00683     /* Check number of timesteps... */
00684     NC_INQ_DIM("time", &nt, 12, 12);
00685
00686     /* Read data... */
00687     ALLOC(help, double,
00688         clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
00689     NC_GET_DOUBLE("h2o2", help, 1);
00690     for (it = 0; it < clim->h2o2_ntime; it++)
00691         for (iz = 0; iz < clim->h2o2_np; iz++)
00692             for (iy = 0; iy < clim->h2o2_nlat; iy++) {
00693                 clim->h2o2[it][iz][iy] =
00694                     help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
00695                 h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
00696                 h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
00697             }
00698     free(help);
00699
00700     /* Close netCDF file... */
00701     NC(nc_close(ncid));
00702
00703     /* Write info... */
00704     LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
00705     LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00706         clim->h2o2_time[0], clim->h2o2_time[1],
00707         clim->h2o2_time[clim->h2o2_ntime - 1]);
00708     LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
00709     LOG(2, "Altitude levels: %g, %g ... %g km",
00710         Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
00711         Z(clim->h2o2_p[clim->h2o2_np - 1]));
00712     LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
00713         clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
00714     LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
00715     LOG(2, "Latitudes: %g, %g ... %g deg",
00716         clim->h2o2_lat[0], clim->h2o2_lat[1],
00717         clim->h2o2_lat[clim->h2o2_nlat - 1]);
00718     LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);

```

```

00719 }
00720
00721 /*****
00722
00723 double clim_tropo(
00724     clim_t * clim,
00725     double t,
00726     double lat) {
00727
00728     /* Get seconds since begin of year... */
00729     double sec = FMOD(t, 365.25 * 86400.);
00730     while (sec < 0)
00731         sec += 365.25 * 86400.;
00732
00733     /* Get indices... */
00734     int isec = locate_irr(clim->tropo_time, clim->tropo_ntime, sec);
00735     int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);
00736
00737     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
00738     double p0 = LIN(clim->tropo_lat[ilat],
00739                     clim->tropo[isec][ilat],
00740                     clim->tropo_lat[ilat + 1],
00741                     clim->tropo[isec][ilat + 1], lat);
00742     double p1 = LIN(clim->tropo_lat[ilat],
00743                     clim->tropo[isec + 1][ilat],
00744                     clim->tropo_lat[ilat + 1],
00745                     clim->tropo[isec + 1][ilat + 1], lat);
00746     return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
00747 }
00748
00749 *****/
00750
00751 void clim_tropo_init(
00752     clim_t * clim) {
00753
00754     /* Write info... */
00755     LOG(1, "Initialize tropopause data...");
00756
00757     clim->tropo_ntime = 12;
00758     double tropo_time[12] = {
00759         1209600.00, 3888000.00, 6393600.00,
00760         9072000.00, 11664000.00, 14342400.00,
00761         16934400.00, 19612800.00, 22291200.00,
00762         24883200.00, 27561600.00, 30153600.00
00763     };
00764     memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));
00765
00766     clim->tropo_nlat = 73;
00767     double tropo_lat[73] = {
00768         -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00769         -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00770         -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00771         -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00772         15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00773         45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00774         75, 77.5, 80, 82.5, 85, 87.5, 90
00775     };
00776     memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));
00777
00778     double tropo[12][73] = {
00779         {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00780          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00781          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00782          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00783          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00784          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00785          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00786          275.3, 275.6, 275.4, 274.1, 273.5},
00787         {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00788          300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00789          150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00790          98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00791          98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00792          220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00793          284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00794          287.5, 286.2, 285.8},
00795         {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00796          297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00797          161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00798          100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00799          99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00800          186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00801          279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00802          304.3, 304.9, 306, 306.6, 306.2, 306},
00803         {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00804          290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00805          195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,

```

```

00806     102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00807     99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00808     148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00809     263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00810     315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00811 {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00812     260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00813     205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00814     101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00815     102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00816     165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00817     273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00818     325.3, 325.8, 325.8},
00819 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00820     222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00821     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00822     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00823     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00824     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00825     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00826     308.5, 312.2, 313.1, 313.3},
00827 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00828     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00829     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00830     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00831     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00832     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00833     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00834     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00835 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00836     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00837     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00838     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00839     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
00840     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00841     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00842     278.2, 282.6, 287.4, 290.9, 292.5, 293},
00843 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00844     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00845     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00846     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00847     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00848     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00849     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00850     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00851 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00852     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00853     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00854     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00855     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00856     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00857     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00858     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00859     305.1},
00860 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00861     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00862     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00863     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00864     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00865     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00866     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00867     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00868 {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00869     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00870     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00871     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00872     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00873     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00874     280.3, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00875     281.7, 281.1, 281.2}
00876 };
00877 memcpy(clim->tropo, tropo, sizeof(clim->tropo));
00878
00879 /* Get range... */
00880 double tropomin = 1e99, tropomax = -1e99;
00881 for (int it = 0; it < clim->tropo_ntime; it++)
00882     for (int iy = 0; iy < clim->tropo_nlat; iy++) {
00883         tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
00884         tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
00885     }
00886
00887 /* Write info... */
00888 LOG(2, "Number of time steps: %d", clim->tropo_ntime);
00889 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00890     clim->tropo_time[0], clim->tropo_time[1],
00891     clim->tropo_time[clim->tropo_ntime - 1]);
00892 LOG(2, "Number of latitudes: %d", clim->tropo_nlat);

```

```

00893 LOG(2, "Latitudes: %g, %g ... %g deg",
00894     clim->tropo_lat[0], clim->tropo_lat[1],
00895     clim->tropo_lat[clim->tropo_nlat - 1]);
00896 LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
00897     Z(tropomin));
00898 LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
00899 }
00900
00901 /*****
00902
00903 void compress_pack(
00904     char *varname,
00905     float *array,
00906     size_t nxy,
00907     size_t nz,
00908     int decompress,
00909     FILE * inout) {
00910
00911     double min[EP], max[EP], off[EP], scl[EP];
00912
00913     unsigned short *sarray;
00914
00915     /* Allocate... */
00916     ALLOC(sarray, unsigned short,
00917         nxy * nz);
00918
00919     /* Read compressed stream and decompress array... */
00920     if (decompress) {
00921
00922         /* Write info... */
00923         LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
00924             varname, 100. * sizeof(unsigned short) / sizeof(float));
00925
00926         /* Read data... */
00927         FREAD(&scl, double,
00928             nz,
00929             inout);
00930         FREAD(&off, double,
00931             nz,
00932             inout);
00933         FREAD(sarray, unsigned short,
00934             nxy * nz,
00935             inout);
00936
00937         /* Convert to float... */
00938 #pragma omp parallel for default(shared)
00939         for (size_t ixy = 0; ixy < nxy; ixy++)
00940             for (size_t iz = 0; iz < nz; iz++)
00941                 array[ixy * nz + iz]
00942                     = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
00943     }
00944
00945     /* Compress array and output compressed stream... */
00946     else {
00947
00948         /* Write info... */
00949         LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
00950             varname, 100. * sizeof(unsigned short) / sizeof(float));
00951
00952         /* Get range... */
00953         for (size_t iz = 0; iz < nz; iz++) {
00954             min[iz] = array[iz];
00955             max[iz] = array[iz];
00956         }
00957         for (size_t ixy = 1; ixy < nxy; ixy++)
00958             for (size_t iz = 0; iz < nz; iz++) {
00959                 if (array[ixy * nz + iz] < min[iz])
00960                     min[iz] = array[ixy * nz + iz];
00961                 if (array[ixy * nz + iz] > max[iz])
00962                     max[iz] = array[ixy * nz + iz];
00963             }
00964
00965         /* Get offset and scaling factor... */
00966         for (size_t iz = 0; iz < nz; iz++) {
00967             scl[iz] = (max[iz] - min[iz]) / 65533.;
00968             off[iz] = min[iz];
00969         }
00970
00971         /* Convert to short... */
00972 #pragma omp parallel for default(shared)
00973         for (size_t ixy = 0; ixy < nxy; ixy++)
00974             for (size_t iz = 0; iz < nz; iz++)
00975                 if (scl[iz] != 0)
00976                     sarray[ixy * nz + iz] = (unsigned short)
00977                         ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
00978                 else
00979                     sarray[ixy * nz + iz] = 0;

```



```

00980
00981     /* Write data... */
00982     FWRITE(&scl, double,
00983           nz,
00984           inout);
00985     FWRITE(&off, double,
00986           nz,
00987           inout);
00988     FWRITE(sarray, unsigned short,
00989           nxy * nz,
00990           inout);
00991 }
00992
00993 /* Free... */
00994 free(sarray);
00995 }
00996
00997 /*****
00998
00999 #ifndef ZFP
01000 void compress_zfp(
01001     char *varname,
01002     float *array,
01003     int nx,
01004     int ny,
01005     int nz,
01006     int precision,
01007     double tolerance,
01008     int decompress,
01009     FILE * inout) {
01010
01011     zfp_type type;           /* array scalar type */
01012     zfp_field *field;        /* array meta data */
01013     zfp_stream *zfp;        /* compressed stream */
01014     void *buffer;           /* storage for compressed stream */
01015     size_t bufsize;         /* byte size of compressed buffer */
01016     bitstream *stream;      /* bit stream to write to or read from */
01017     size_t zfpsize;         /* byte size of compressed stream */
01018
01019     /* Allocate meta data for the 3D array a[nz][ny][nx]... */
01020     type = zfp_type_float;
01021     field = zfp_field_3d(array, type, (uint) nx, (uint) ny, (uint) nz);
01022
01023     /* Allocate meta data for a compressed stream... */
01024     zfp = zfp_stream_open(NULL);
01025
01026     /* Set compression mode... */
01027     int actual_prec = 0;
01028     double actual_tol = 0;
01029     if (precision > 0)
01030         actual_prec = (int) zfp_stream_set_precision(zfp, (uint) precision);
01031     else if (tolerance > 0)
01032         actual_tol = zfp_stream_set_accuracy(zfp, tolerance);
01033     else
01034         ERRMSG("Set precision or tolerance!");
01035
01036     /* Allocate buffer for compressed data... */
01037     bufsize = zfp_stream_maximum_size(zfp, field);
01038     buffer = malloc(bufsize);
01039
01040     /* Associate bit stream with allocated buffer... */
01041     stream = stream_open(buffer, bufsize);
01042     zfp_stream_set_bit_stream(zfp, stream);
01043     zfp_stream_rewind(zfp);
01044
01045     /* Read compressed stream and decompress array... */
01046     if (decompress) {
01047         FREAD(&zfpsize, size_t,
01048             1,
01049             inout);
01050         if (fread(buffer, 1, zfpsize, inout) != zfpsize)
01051             ERRMSG("Error while reading zfp data!");
01052         if (!zfp_decompress(zfp, field)) {
01053             ERRMSG("Decompression failed!");
01054         }
01055         LOG(2, "Read 3-D variable: %s "
01056             "(zfp, PREC= %d, TOL= %g, RATIO= %g %%)",
01057             varname, actual_prec, actual_tol,
01058             (100. * (double) zfpsize) / (double) (nx * ny * nz));
01059     }
01060
01061     /* Compress array and output compressed stream... */
01062     else {
01063         zfpsize = zfp_compress(zfp, field);
01064         if (!zfpsize) {
01065             ERRMSG("Compression failed!");
01066         } else {

```

```

01067     FWRITE(&zfp_size, size_t,
01068           1,
01069           inout);
01070     if (fwrite(buffer, 1, zfp_size, inout) != zfp_size)
01071         ERRMSG("Error while writing zfp data!");
01072 }
01073 LOG(2, "Write 3-D variable: %s "
01074      "(zfp, PREC= %d, TOL= %g, RATIO= %g %%)",
01075      varname, actual_prec, actual_tol,
01076      (100. * (double) zfp_size) / (double) (nx * ny * nz));
01077 }
01078
01079 /* Free... */
01080 zfp_field_free(field);
01081 zfp_stream_close(zfp);
01082 stream_close(stream);
01083 free(buffer);
01084 }
01085 #endif
01086
01087 /*****
01088
01089 #ifdef ZSTD
01090 void compress_zstd(
01091     char *varname,
01092     float *array,
01093     size_t n,
01094     int decompress,
01095     FILE * inout) {
01096
01097     /* Get buffer sizes... */
01098     size_t uncompressLen = n * sizeof(float);
01099     size_t comprLen = ZSTD_compressBound(uncompressLen);
01100     size_t compsize;
01101
01102     /* Allocate... */
01103     char *compr = (char *) calloc((uint) comprLen, 1);
01104     char *uncompr = (char *) array;
01105
01106     /* Read compressed stream and decompress array... */
01107     if (decompress) {
01108         FREAD(&comprLen, size_t,
01109             1,
01110             inout);
01111         if (fread(compr, 1, comprLen, inout) != comprLen)
01112             ERRMSG("Error while reading zstd data!");
01113         compsize = ZSTD_decompress(uncompr, uncompressLen, compr, comprLen);
01114         if (ZSTD_isError(compsize)) {
01115             ERRMSG("Decompression failed!");
01116         }
01117         LOG(2, "Read 3-D variable: %s (zstd, RATIO= %g %%)",
01118             varname, (100. * (double) comprLen) / (double) uncompressLen);
01119     }
01120
01121     /* Compress array and output compressed stream... */
01122     else {
01123         compsize = ZSTD_compress(compr, comprLen, uncompr, uncompressLen, 0);
01124         if (ZSTD_isError(compsize)) {
01125             ERRMSG("Compression failed!");
01126         } else {
01127             FWRITE(&compsize, size_t,
01128                 1,
01129                 inout);
01130             if (fwrite(compr, 1, compsize, inout) != compsize)
01131                 ERRMSG("Error while writing zstd data!");
01132         }
01133         LOG(2, "Write 3-D variable: %s (zstd, RATIO= %g %%)",
01134             varname, (100. * (double) compsize) / (double) uncompressLen);
01135     }
01136
01137     /* Free... */
01138     free(compr);
01139 }
01140 #endif
01141
01142 /*****
01143
01144 void day2doy(
01145     int year,
01146     int mon,
01147     int day,
01148     int *doy) {
01149
01150     const int
01151         d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01152         d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01153

```

```

01154  /* Get day of year... */
01155  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01156      *doy = d0l[mon - 1] + day - 1;
01157  else
01158      *doy = d0[mon - 1] + day - 1;
01159  }
01160
01161  /*****
01162
01163  void doy2day(
01164      int year,
01165      int doy,
01166      int *mon,
01167      int *day) {
01168
01169      const int
01170          d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },
01171          d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01172
01173      int i;
01174
01175      /* Get month and day... */
01176      if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01177          for (i = 11; i > 0; i--)
01178              if (d0l[i] <= doy)
01179                  break;
01180          *mon = i + 1;
01181          *day = doy - d0l[i] + 1;
01182      } else {
01183          for (i = 11; i > 0; i--)
01184              if (d0[i] <= doy)
01185                  break;
01186          *mon = i + 1;
01187          *day = doy - d0[i] + 1;
01188      }
01189  }
01190
01191  /*****
01192
01193  void geo2cart(
01194      double z,
01195      double lon,
01196      double lat,
01197      double *x) {
01198
01199      double radius = z + RE;
01200      x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01201      x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01202      x[2] = radius * sin(lat / 180. * M_PI);
01203  }
01204
01205  /*****
01206
01207  void get_met(
01208      ctl_t *ctl,
01209      clim_t *clim,
01210      double t,
01211      met_t **met0,
01212      met_t **met1) {
01213
01214      static int init;
01215
01216      met_t *mets;
01217
01218      char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01219
01220      /* Set timer... */
01221      SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01222
01223      /* Init... */
01224      if (t == ctl->t_start || !init) {
01225          init = 1;
01226
01227          /* Read meteo data... */
01228          get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
01229                      ctl->metbase, ctl->dt_met, filename);
01229          if (!read_met(filename, ctl, clim, *met0))
01230              ERRMSG("Cannot open file!");
01231
01232          get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
01233                      ctl->metbase, ctl->dt_met, filename);
01234          if (!read_met(filename, ctl, clim, *met1))
01235              ERRMSG("Cannot open file!");
01236
01237          /* Update GPU... */
01238          #ifdef _OPENACC
01239              met_t *met0up = *met0;

```

```

01241     met_t *metlup = *metl;
01242 #ifdef ASYNCIO
01243 #pragma acc update device(met0up[:1],metlup[:1]) async(5)
01244 #else
01245 #pragma acc update device(met0up[:1],metlup[:1])
01246 #endif
01247 #endif
01248
01249 /* Caching... */
01250 if (ctl->met_cache && t != ctl->t_stop) {
01251     get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
01252                 ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
01253     sprintf(cmd, "cat %s > /dev/null &", cachefile);
01254     LOG(1, "Caching: %s", cachefile);
01255     if (system(cmd) != 0)
01256         WARN("Caching command failed!");
01257 }
01258 }
01259
01260 /* Read new data for forward trajectories... */
01261 if (t > (*metl)->time) {
01262
01263     /* Pointer swap... */
01264     mets = *metl;
01265     *metl = *met0;
01266     *met0 = mets;
01267
01268     /* Read new meteo data... */
01269     get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
01270     if (!read_met(filename, ctl, clim, *metl))
01271         ERRMSG("Cannot open file!");
01272
01273     /* Update GPU... */
01274 #ifdef _OPENACC
01275     met_t *metlup = *metl;
01276 #ifdef ASYNCIO
01277 #pragma acc update device(metlup[:1]) async(5)
01278 #else
01279 #pragma acc update device(metlup[:1])
01280 #endif
01281 #endif
01282
01283     /* Caching... */
01284     if (ctl->met_cache && t != ctl->t_stop) {
01285         get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
01286                     cachefile);
01287         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01288         LOG(1, "Caching: %s", cachefile);
01289         if (system(cmd) != 0)
01290             WARN("Caching command failed!");
01291     }
01292 }
01293
01294 /* Read new data for backward trajectories... */
01295 if (t < (*met0)->time) {
01296
01297     /* Pointer swap... */
01298     mets = *metl;
01299     *metl = *met0;
01300     *met0 = mets;
01301
01302     /* Read new meteo data... */
01303     get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
01304     if (!read_met(filename, ctl, clim, *met0))
01305         ERRMSG("Cannot open file!");
01306
01307     /* Update GPU... */
01308 #ifdef _OPENACC
01309     met_t *met0up = *met0;
01310 #ifdef ASYNCIO
01311 #pragma acc update device(met0up[:1]) async(5)
01312 #else
01313 #pragma acc update device(met0up[:1])
01314 #endif
01315 #endif
01316
01317     /* Caching... */
01318     if (ctl->met_cache && t != ctl->t_stop) {
01319         get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
01320                     cachefile);
01321         sprintf(cmd, "cat %s > /dev/null &", cachefile);
01322         LOG(1, "Caching: %s", cachefile);
01323         if (system(cmd) != 0)
01324             WARN("Caching command failed!");
01325     }
01326 }
01327
01328 /* Check that grids are consistent... */
01329 if ((*met0)->nx != 0 && (*metl)->nx != 0) {

```

```

01328     if ((*met0)->nx != (*met1)->nx
01329         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01330         ERRMSG("Meteo grid dimensions do not match!");
01331     for (int ix = 0; ix < (*met0)->nx; ix++)
01332         if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01333             ERRMSG("Meteo grid longitudes do not match!");
01334     for (int iy = 0; iy < (*met0)->ny; iy++)
01335         if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01336             ERRMSG("Meteo grid latitudes do not match!");
01337     for (int ip = 0; ip < (*met0)->np; ip++)
01338         if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01339             ERRMSG("Meteo grid pressure levels do not match!");
01340 }
01341 }
01342
01343 /*****
01344
01345 void get_met_help(
01346     ctl_t * ctl,
01347     double t,
01348     int direct,
01349     char *metbase,
01350     double dt_met,
01351     char *filename) {
01352
01353     char repl[LEN];
01354
01355     double t6, r;
01356
01357     int year, mon, day, hour, min, sec;
01358
01359     /* Round time to fixed intervals... */
01360     if (direct == -1)
01361         t6 = floor(t / dt_met) * dt_met;
01362     else
01363         t6 = ceil(t / dt_met) * dt_met;
01364
01365     /* Decode time... */
01366     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01367
01368     /* Set filename of MPTRAC meteo files... */
01369     if (ctl->clams_met_data == 0) {
01370         if (ctl->met_type == 0)
01371             sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01372         else if (ctl->met_type == 1)
01373             sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
01374         else if (ctl->met_type == 2)
01375             sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
01376         else if (ctl->met_type == 3)
01377             sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
01378         else if (ctl->met_type == 4)
01379             sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", metbase);
01380         sprintf(repl, "%d", year);
01381         get_met_replace(filename, "YYYY", repl);
01382         sprintf(repl, "%02d", mon);
01383         get_met_replace(filename, "MM", repl);
01384         sprintf(repl, "%02d", day);
01385         get_met_replace(filename, "DD", repl);
01386         sprintf(repl, "%02d", hour);
01387         get_met_replace(filename, "HH", repl);
01388     }
01389
01390     /* Set filename of CLaMS meteo files... */
01391     else {
01392         sprintf(filename, "%s_YYMMDDHH.nc", metbase);
01393         sprintf(repl, "%d", year);
01394         get_met_replace(filename, "YYYY", repl);
01395         sprintf(repl, "%d", year % 100);
01396         get_met_replace(filename, "YY", repl);
01397         sprintf(repl, "%02d", mon);
01398         get_met_replace(filename, "MM", repl);
01399         sprintf(repl, "%02d", day);
01400         get_met_replace(filename, "DD", repl);
01401         sprintf(repl, "%02d", hour);
01402         get_met_replace(filename, "HH", repl);
01403     }
01404 }
01405
01406 /*****
01407
01408 void get_met_replace(
01409     char *orig,
01410     char *search,
01411     char *repl) {
01412
01413     char buffer[LEN];
01414

```

```

01415  /* Iterate... */
01416  for (int i = 0; i < 3; i++) {
01417
01418      /* Replace sub-string... */
01419      char *ch;
01420      if (!(ch = strstr(orig, search)))
01421          return;
01422      strncpy(buffer, orig, (size_t) (ch - orig));
01423      buffer[ch - orig] = 0;
01424      sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01425      orig[0] = 0;
01426      strcpy(orig, buffer);
01427  }
01428 }
01429
01430 /*****
01431
01432 void intpol_met_space_3d(
01433     met_t * met,
01434     float array[EX][EY][EP],
01435     double p,
01436     double lon,
01437     double lat,
01438     double *var,
01439     int *ci,
01440     double *cw,
01441     int init) {
01442
01443     /* Initialize interpolation... */
01444     if (init) {
01445
01446         /* Check longitude... */
01447         if (met->lon[met->nx - 1] > 180 && lon < 0)
01448             lon += 360;
01449
01450         /* Get interpolation indices... */
01451         ci[0] = locate_irr(met->p, met->np, p);
01452         ci[1] = locate_reg(met->lon, met->nx, lon);
01453         ci[2] = locate_reg(met->lat, met->ny, lat);
01454
01455         /* Get interpolation weights... */
01456         cw[0] = (met->p[ci[0] + 1] - p)
01457             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01458         cw[1] = (met->lon[ci[1] + 1] - lon)
01459             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01460         cw[2] = (met->lat[ci[2] + 1] - lat)
01461             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01462     }
01463
01464     /* Interpolate vertically... */
01465     double aux00 =
01466         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01467         + array[ci[1]][ci[2]][ci[0] + 1];
01468     double aux01 =
01469         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01470             array[ci[1]][ci[2] + 1][ci[0] + 1])
01471         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01472     double aux10 =
01473         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01474             array[ci[1] + 1][ci[2]][ci[0] + 1])
01475         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01476     double aux11 =
01477         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01478             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01479         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01480
01481     /* Interpolate horizontally... */
01482     aux00 = cw[2] * (aux00 - aux01) + aux01;
01483     aux11 = cw[2] * (aux10 - aux11) + aux11;
01484     *var = cw[1] * (aux00 - aux11) + aux11;
01485 }
01486
01487 /*****
01488
01489 void intpol_met_space_2d(
01490     met_t * met,
01491     float array[EX][EY],
01492     double lon,
01493     double lat,
01494     double *var,
01495     int *ci,
01496     double *cw,
01497     int init) {
01498
01499     /* Initialize interpolation... */
01500     if (init) {
01501

```

```

01502     /* Check longitude... */
01503     if (met->lon[met->nx - 1] > 180 && lon < 0)
01504         lon += 360;
01505
01506     /* Get interpolation indices... */
01507     ci[1] = locate_reg(met->lon, met->nx, lon);
01508     ci[2] = locate_reg(met->lat, met->ny, lat);
01509
01510     /* Get interpolation weights... */
01511     cw[1] = (met->lon[ci[1] + 1] - lon)
01512         / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01513     cw[2] = (met->lat[ci[2] + 1] - lat)
01514         / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01515 }
01516
01517 /* Set variables... */
01518 double aux00 = array[ci[1]][ci[2]];
01519 double aux01 = array[ci[1]][ci[2] + 1];
01520 double aux10 = array[ci[1] + 1][ci[2]];
01521 double aux11 = array[ci[1] + 1][ci[2] + 1];
01522
01523 /* Interpolate horizontally... */
01524 if (isfinite(aux00) && isfinite(aux01)
01525     && isfinite(aux10) && isfinite(aux11)) {
01526     aux00 = cw[2] * (aux00 - aux01) + aux01;
01527     aux11 = cw[2] * (aux10 - aux11) + aux11;
01528     *var = cw[1] * (aux00 - aux11) + aux11;
01529 } else {
01530     if (cw[2] < 0.5) {
01531         if (cw[1] < 0.5)
01532             *var = aux11;
01533         else
01534             *var = aux01;
01535     } else {
01536         if (cw[1] < 0.5)
01537             *var = aux10;
01538         else
01539             *var = aux00;
01540     }
01541 }
01542 }
01543
01544 /*****
01545 #ifdef UVW
01546 void intpol_met_space_uvw(
01547     met_t * met,
01548     double p,
01549     double lon,
01550     double lat,
01551     double *u,
01552     double *v,
01553     double *w,
01554     int *ci,
01555     double *cw,
01556     int init) {
01557
01558     /* Initialize interpolation... */
01559     if (init) {
01560
01561         /* Check longitude... */
01562         if (met->lon[met->nx - 1] > 180 && lon < 0)
01563             lon += 360;
01564
01565         /* Get interpolation indices... */
01566         ci[0] = locate_irr(met->p, met->np, p);
01567         ci[1] = locate_reg(met->lon, met->nx, lon);
01568         ci[2] = locate_reg(met->lat, met->ny, lat);
01569
01570         /* Get interpolation weights... */
01571         cw[0] = (met->p[ci[0] + 1] - p)
01572             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01573         cw[1] = (met->lon[ci[1] + 1] - lon)
01574             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01575         cw[2] = (met->lat[ci[2] + 1] - lat)
01576             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01577     }
01578
01579     /* Interpolate vertically... */
01580     double u00 =
01581         cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][0] -
01582             met->uvw[ci[1]][ci[2]][ci[0] + 1][0])
01583         + met->uvw[ci[1]][ci[2]][ci[0] + 1][0];
01584     double u01 =
01585         cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][0] -
01586             met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][0])
01587         + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][0];

```

```

01589 double u10 =
01590     cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][0] -
01591             met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][0])
01592     + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][0];
01593 double u11 =
01594     cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][0] -
01595             met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][0])
01596     + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][0];
01597
01598 double v00 =
01599     cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][1] -
01600             met->uvw[ci[1]][ci[2]][ci[0] + 1][1])
01601     + met->uvw[ci[1]][ci[2]][ci[0] + 1][1];
01602 double v01 =
01603     cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][1] -
01604             met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][1])
01605     + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][1];
01606 double v10 =
01607     cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][1] -
01608             met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][1])
01609     + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][1];
01610 double v11 =
01611     cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][1] -
01612             met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][1])
01613     + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][1];
01614
01615 double w00 =
01616     cw[0] * (met->uvw[ci[1]][ci[2]][ci[0]][2] -
01617             met->uvw[ci[1]][ci[2]][ci[0] + 1][2])
01618     + met->uvw[ci[1]][ci[2]][ci[0] + 1][2];
01619 double w01 =
01620     cw[0] * (met->uvw[ci[1]][ci[2] + 1][ci[0]][2] -
01621             met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][2])
01622     + met->uvw[ci[1]][ci[2] + 1][ci[0] + 1][2];
01623 double w10 =
01624     cw[0] * (met->uvw[ci[1] + 1][ci[2]][ci[0]][2] -
01625             met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][2])
01626     + met->uvw[ci[1] + 1][ci[2]][ci[0] + 1][2];
01627 double w11 =
01628     cw[0] * (met->uvw[ci[1] + 1][ci[2] + 1][ci[0]][2] -
01629             met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][2])
01630     + met->uvw[ci[1] + 1][ci[2] + 1][ci[0] + 1][2];
01631
01632 /* Interpolate horizontally... */
01633 u00 = cw[2] * (u00 - u01) + u01;
01634 u11 = cw[2] * (u10 - u11) + u11;
01635 *u = cw[1] * (u00 - u11) + u11;
01636
01637 v00 = cw[2] * (v00 - v01) + v01;
01638 v11 = cw[2] * (v10 - v11) + v11;
01639 *v = cw[1] * (v00 - v11) + v11;
01640
01641 w00 = cw[2] * (w00 - w01) + w01;
01642 w11 = cw[2] * (w10 - w11) + w11;
01643 *w = cw[1] * (w00 - w11) + w11;
01644 }
01645 #endif
01646
01647 /*****
01648
01649 void intpol_met_time_3d(
01650     met_t * met0,
01651     float array0[EX][EY][EP],
01652     met_t * met1,
01653     float array1[EX][EY][EP],
01654     double ts,
01655     double p,
01656     double lon,
01657     double lat,
01658     double *var,
01659     int *ci,
01660     double *cw,
01661     int init) {
01662
01663     double var0, var1, wt;
01664
01665     /* Spatial interpolation... */
01666     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01667     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01668
01669     /* Get weighting factor... */
01670     wt = (met1->time - ts) / (met1->time - met0->time);
01671
01672     /* Interpolate... */
01673     *var = wt * (var0 - var1) + var1;
01674 }
01675

```



```

01676 /*****
01677
01678 void intpol_met_time_2d(
01679     met_t * met0,
01680     float array0[EX][EY],
01681     met_t * met1,
01682     float array1[EX][EY],
01683     double ts,
01684     double lon,
01685     double lat,
01686     double *var,
01687     int *ci,
01688     double *cw,
01689     int init) {
01690
01691     double var0, var1, wt;
01692
01693     /* Spatial interpolation... */
01694     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01695     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01696
01697     /* Get weighting factor... */
01698     wt = (met1->time - ts) / (met1->time - met0->time);
01699
01700     /* Interpolate... */
01701     if (isfinite(var0) && isfinite(var1))
01702         *var = wt * (var0 - var1) + var1;
01703     else if (wt < 0.5)
01704         *var = var1;
01705     else
01706         *var = var0;
01707 }
01708
01709 /*****
01710
01711 #ifdef UVW
01712 void intpol_met_time_uvw(
01713     met_t * met0,
01714     met_t * met1,
01715     double ts,
01716     double p,
01717     double lon,
01718     double lat,
01719     double *u,
01720     double *v,
01721     double *w) {
01722
01723     double u0, u1, v0, v1, w0, w1, wt;
01724
01725     /* Spatial interpolation... */
01726     INTPOL_INIT;
01727     intpol_met_space_uvw(met0, p, lon, lat, &u0, &v0, &w0, ci, cw, 1);
01728     intpol_met_space_uvw(met1, p, lon, lat, &u1, &v1, &w1, ci, cw, 0);
01729
01730     /* Get weighting factor... */
01731     wt = (met1->time - ts) / (met1->time - met0->time);
01732
01733     /* Interpolate... */
01734     *u = wt * (u0 - u1) + u1;
01735     *v = wt * (v0 - v1) + v1;
01736     *w = wt * (w0 - w1) + w1;
01737 }
01738 #endif
01739
01740 /*****
01741
01742 void jsec2time(
01743     double jsec,
01744     int *year,
01745     int *mon,
01746     int *day,
01747     int *hour,
01748     int *min,
01749     int *sec,
01750     double *remain) {
01751
01752     struct tm t0, *t1;
01753
01754     t0.tm_year = 100;
01755     t0.tm_mon = 0;
01756     t0.tm_mday = 1;
01757     t0.tm_hour = 0;
01758     t0.tm_min = 0;
01759     t0.tm_sec = 0;
01760
01761     time_t jsec0 = (time_t) jsec + timegm(&t0);
01762     t1 = gmtime(&jsec0);

```

```

01763
01764 *year = t1->tm_year + 1900;
01765 *mon = t1->tm_mon + 1;
01766 *day = t1->tm_mday;
01767 *hour = t1->tm_hour;
01768 *min = t1->tm_min;
01769 *sec = t1->tm_sec;
01770 *remain = jsec - floor(jsec);
01771 }
01772
01773 /*****
01774
01775 double lapse_rate(
01776     double t,
01777     double h2o) {
01778
01779     /*
01780      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01781      and water vapor volume mixing ratio [1].
01782
01783      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01784     */
01785
01786     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01787
01788     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01789 }
01790
01791 /*****
01792
01793 int locate_irr(
01794     double *xx,
01795     int n,
01796     double x) {
01797
01798     int ilo = 0;
01799     int ihi = n - 1;
01800     int i = (ihi + ilo) >> 1;
01801
01802     if (xx[i] < xx[i + 1])
01803         while (ihi > ilo + 1) {
01804             i = (ihi + ilo) >> 1;
01805             if (xx[i] > x)
01806                 ihi = i;
01807             else
01808                 ilo = i;
01809         } else
01810             while (ihi > ilo + 1) {
01811                 i = (ihi + ilo) >> 1;
01812                 if (xx[i] <= x)
01813                     ihi = i;
01814                 else
01815                     ilo = i;
01816             }
01817
01818     return ilo;
01819 }
01820
01821 /*****
01822
01823 int locate_reg(
01824     double *xx,
01825     int n,
01826     double x) {
01827
01828     /* Calculate index... */
01829     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01830
01831     /* Check range... */
01832     if (i < 0)
01833         return 0;
01834     else if (i > n - 2)
01835         return n - 2;
01836     else
01837         return i;
01838 }
01839
01840 /*****
01841
01842 double nat_temperature(
01843     double p,
01844     double h2o,
01845     double hno3) {
01846
01847     /* Check water vapor vmr... */
01848     h2o = GSL_MAX(h2o, 0.1e-6);
01849

```

```

01850  /* Calculate T_NAT... */
01851  double p_hno3 = hno3 * p / 1.333224;
01852  double p_h2o = h2o * p / 1.333224;
01853  double a = 0.009179 - 0.00088 * log10(p_h2o);
01854  double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01855  double c = -11397.0 / a;
01856  double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
01857  double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01858  if (x2 > 0)
01859      tnat = x2;
01860
01861  return tnat;
01862 }
01863
01864 /*****
01865 void quicksort(
01866     double arr[],
01867     int brr[],
01868     int low,
01869     int high) {
01870
01871     if (low < high) {
01872         int pi = quicksort_partition(arr, brr, low, high);
01873
01874         #pragma omp task firstprivate(arr,brr,low,pi)
01875         {
01876             quicksort(arr, brr, low, pi - 1);
01877         }
01878
01879         // #pragma omp task firstprivate(arr,brr,high,pi)
01880         {
01881             quicksort(arr, brr, pi + 1, high);
01882         }
01883     }
01884 }
01885
01886 int quicksort_partition(
01887     double arr[],
01888     int brr[],
01889     int low,
01890     int high) {
01891
01892     double pivot = arr[high];
01893     int i = (low - 1);
01894
01895     for (int j = low; j <= high - 1; j++)
01896         if (arr[j] <= pivot) {
01897             i++;
01898             SWAP(arr[i], arr[j], double);
01899             SWAP(brr[i], brr[j], int);
01900         }
01901     SWAP(arr[high], arr[i + 1], double);
01902     SWAP(brr[high], brr[i + 1], int);
01903
01904     return (i + 1);
01905 }
01906
01907 /*****
01908 int read_atm(
01909     const char *filename,
01910     ctl_t * ctl,
01911     atm_t * atm) {
01912
01913     int result;
01914
01915     /* Set timer... */
01916     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
01917
01918     /* Init... */
01919     atm->np = 0;
01920
01921     /* Write info... */
01922     LOG(1, "Read atmospheric data: %s", filename);
01923
01924     /* Read ASCII data... */
01925     if (ctl->atm_type == 0)
01926         result = read_atm_asc(filename, ctl, atm);
01927
01928     /* Read binary data... */
01929     else if (ctl->atm_type == 1)
01930         result = read_atm_bin(filename, ctl, atm);
01931
01932     /* Read netCDF data... */

```

```

01937     else if (ctl->atm_type == 2)
01938         result = read_atm_nc(filename, ctl, atm);
01939
01940     /* Read CLaMS data... */
01941     else if (ctl->atm_type == 3)
01942         result = read_atm_clams(filename, ctl, atm);
01943
01944     /* Error... */
01945     else
01946         ERRMSG("Atmospheric data type not supported!");
01947
01948     /* Check result... */
01949     if (result != 1)
01950         return 0;
01951
01952     /* Check number of air parcels... */
01953     if (atm->np < 1)
01954         ERRMSG("Can not read any data!");
01955
01956     /* Write info... */
01957     double mini, maxi;
01958     LOG(2, "Number of particles: %d", atm->np);
01959     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
01960     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
01961     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
01962     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
01963     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
01964     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
01965     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
01966     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
01967     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
01968     for (int iq = 0; iq < ctl->nq; iq++) {
01969         char msg[LEN];
01970         sprintf(msg, "Quantity %s range: %s ... %s %s",
01971                 ctl->qnt_name[iq], ctl->qnt_format[iq],
01972                 ctl->qnt_format[iq], ctl->qnt_unit[iq]);
01973         gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
01974         LOG(2, msg, mini, maxi);
01975     }
01976
01977     /* Return success... */
01978     return 1;
01979 }
01980
01981 /*****
01982
01983 int read_atm_asc(
01984     const char *filename,
01985     ctl_t * ctl,
01986     atm_t * atm) {
01987
01988     FILE *in;
01989
01990     /* Open file... */
01991     if (!(in = fopen(filename, "r"))) {
01992         WARN("Cannot open file!");
01993         return 0;
01994     }
01995
01996     /* Read line... */
01997     char line[LEN];
01998     while (fgets(line, LEN, in)) {
01999
02000         /* Read data... */
02001         char *tok;
02002         TOK(line, tok, "%lg", atm->time[atm->np]);
02003         TOK(NULL, tok, "%lg", atm->p[atm->np]);
02004         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02005         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02006         for (int iq = 0; iq < ctl->nq; iq++)
02007             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02008
02009         /* Convert altitude to pressure... */
02010         atm->p[atm->np] = P(atm->p[atm->np]);
02011
02012         /* Increment data point counter... */
02013         if ((++atm->np) > NP)
02014             ERRMSG("Too many data points!");
02015     }
02016
02017     /* Close file... */
02018     fclose(in);
02019
02020     /* Return success... */
02021     return 1;
02022 }
02023

```

```

02024 /*****
02025
02026 int read_atm_bin(
02027     const char *filename,
02028     ctl_t * ctl,
02029     atm_t * atm) {
02030
02031     FILE *in;
02032
02033     /* Open file... */
02034     if (!(in = fopen(filename, "r")))
02035         return 0;
02036
02037     /* Check version of binary data... */
02038     int version;
02039     FREAD(&version, int,
02040          1,
02041          in);
02042     if (version != 100)
02043         ERRMSG("Wrong version of binary data!");
02044
02045     /* Read data... */
02046     FREAD(&atm->np, int,
02047          1,
02048          in);
02049     FREAD(atm->time, double,
02050          (size_t) atm->np,
02051          in);
02052     FREAD(atm->p, double,
02053          (size_t) atm->np,
02054          in);
02055     FREAD(atm->lon, double,
02056          (size_t) atm->np,
02057          in);
02058     FREAD(atm->lat, double,
02059          (size_t) atm->np,
02060          in);
02061     for (int iq = 0; iq < ctl->nq; iq++)
02062         FREAD(atm->q[iq], double,
02063              (size_t) atm->np,
02064              in);
02065
02066     /* Read final flag... */
02067     int final;
02068     FREAD(&final, int,
02069          1,
02070          in);
02071     if (final != 999)
02072         ERRMSG("Error while reading binary data!");
02073
02074     /* Close file... */
02075     fclose(in);
02076
02077     /* Return success... */
02078     return 1;
02079 }
02080
02081 /*****
02082
02083 int read_atm_clams(
02084     const char *filename,
02085     ctl_t * ctl,
02086     atm_t * atm) {
02087
02088     int ncid, varid;
02089
02090     /* Open file... */
02091     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02092         return 0;
02093
02094     /* Get dimensions... */
02095     NC_INQ_DIM("NPARTS", &atm->np, 1, NP);
02096
02097     /* Get time... */
02098     if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
02099         NC_GET_VAR_DOUBLE(ncid, varid, atm->time);
02100     } else {
02101         WARN("TIME_INIT not found use time instead!");
02102         double time_init;
02103         NC_GET_DOUBLE("time", &time_init, 1);
02104         for (int ip = 0; ip < atm->np; ip++) {
02105             atm->time[ip] = time_init;
02106         }
02107     }
02108
02109     /* Read zeta coordinate, pressure is optional... */
02110     if (ctl->vert_coord_ap == 1) {

```

```

02111     NC_GET_DOUBLE("ZETA", atm->zeta, 1);
02112     NC_GET_DOUBLE("PRESS", atm->p, 0);
02113 }
02114
02115 /* Read pressure, zeta coordinate is optional... */
02116 else {
02117     NC_GET_DOUBLE("PRESS", atm->p, 1);
02118     NC_GET_DOUBLE("ZETA", atm->zeta, 0);
02119 }
02120
02121 /* Read longitude and latitude... */
02122 NC_GET_DOUBLE("LON", atm->lon, 1);
02123 NC_GET_DOUBLE("LAT", atm->lat, 1);
02124
02125 /* Close file... */
02126 NC(nc_close(ncid));
02127
02128 /* Return success... */
02129 return 1;
02130 }
02131
02132 /*****
02133
02134 int read_atm_nc(
02135     const char *filename,
02136     ctl_t * ctl,
02137     atm_t * atm) {
02138
02139     int ncid, varid;
02140
02141     /* Open file... */
02142     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02143         return 0;
02144
02145     /* Get dimensions... */
02146     NC_INQ_DIM("obs", &atm->np, 1, NP);
02147
02148     /* Read geolocations... */
02149     NC_GET_DOUBLE("time", atm->time, 1);
02150     NC_GET_DOUBLE("press", atm->p, 1);
02151     NC_GET_DOUBLE("lon", atm->lon, 1);
02152     NC_GET_DOUBLE("lat", atm->lat, 1);
02153
02154     /* Read variables... */
02155     for (int iq = 0; iq < ctl->nq; iq++)
02156         NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
02157
02158     /* Close file... */
02159     NC(nc_close(ncid));
02160
02161     /* Return success... */
02162     return 1;
02163 }
02164
02165 /*****
02166
02167 void read_clim(
02168     ctl_t * ctl,
02169     clim_t * clim) {
02170
02171     /* Set timer... */
02172     SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);
02173
02174     /* Init tropopause climatology... */
02175     clim_tropo_init(clim);
02176
02177     /* Init HNO3 climatology... */
02178     clim_hno3_init(clim);
02179
02180     /* Read OH climatology... */
02181     if (ctl->clim_oh_filename[0] != '-')
02182         clim_oh_init(ctl, clim);
02183
02184     /* Read H2O2 climatology... */
02185     if (ctl->clim_h2o2_filename[0] != '-')
02186         clim_h2o2_init(ctl, clim);
02187 }
02188
02189 /*****
02190
02191 void read_ctl(
02192     const char *filename,
02193     int argc,
02194     char *argv[],
02195     ctl_t * ctl) {
02196
02197     /* Set timer... */

```

```

02198     SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02199
02200     /* Write info... */
02201     LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02202         "(executable: %s | version: %s | compiled: %s, %s)\n",
02203         argv[0], VERSION, __DATE__, __TIME__);
02204
02205     /* Initialize quantity indices... */
02206     ctl->qnt_idx = -1;
02207     ctl->qnt_ens = -1;
02208     ctl->qnt_stat = -1;
02209     ctl->qnt_m = -1;
02210     ctl->qnt_vmr = -1;
02211     ctl->qnt_rp = -1;
02212     ctl->qnt_rhop = -1;
02213     ctl->qnt_ps = -1;
02214     ctl->qnt_ts = -1;
02215     ctl->qnt_zs = -1;
02216     ctl->qnt_us = -1;
02217     ctl->qnt_vs = -1;
02218     ctl->qnt_pbl = -1;
02219     ctl->qnt_pt = -1;
02220     ctl->qnt_tt = -1;
02221     ctl->qnt_zt = -1;
02222     ctl->qnt_h2ot = -1;
02223     ctl->qnt_z = -1;
02224     ctl->qnt_p = -1;
02225     ctl->qnt_t = -1;
02226     ctl->qnt_rho = -1;
02227     ctl->qnt_u = -1;
02228     ctl->qnt_v = -1;
02229     ctl->qnt_w = -1;
02230     ctl->qnt_h2o = -1;
02231     ctl->qnt_o3 = -1;
02232     ctl->qnt_lwc = -1;
02233     ctl->qnt_iwc = -1;
02234     ctl->qnt_pct = -1;
02235     ctl->qnt_pcb = -1;
02236     ctl->qnt_cl = -1;
02237     ctl->qnt_plcl = -1;
02238     ctl->qnt_plfc = -1;
02239     ctl->qnt_pel = -1;
02240     ctl->qnt_cape = -1;
02241     ctl->qnt_cin = -1;
02242     ctl->qnt_hno3 = -1;
02243     ctl->qnt_oh = -1;
02244     ctl->qnt_vmrimpl = -1;
02245     ctl->qnt_mloss_oh = -1;
02246     ctl->qnt_mloss_h2o2 = -1;
02247     ctl->qnt_mloss_wet = -1;
02248     ctl->qnt_mloss_dry = -1;
02249     ctl->qnt_mloss_decay = -1;
02250     ctl->qnt_psat = -1;
02251     ctl->qnt_psice = -1;
02252     ctl->qnt_pw = -1;
02253     ctl->qnt_sh = -1;
02254     ctl->qnt_rh = -1;
02255     ctl->qnt_rhice = -1;
02256     ctl->qnt_theta = -1;
02257     ctl->qnt_zeta = -1;
02258     ctl->qnt_tvirt = -1;
02259     ctl->qnt_lapse = -1;
02260     ctl->qnt_vh = -1;
02261     ctl->qnt_vz = -1;
02262     ctl->qnt_pv = -1;
02263     ctl->qnt_tdew = -1;
02264     ctl->qnt_tice = -1;
02265     ctl->qnt_tsts = -1;
02266     ctl->qnt_tnat = -1;
02267
02268     /* Read quantities... */
02269     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02270     if (ctl->nq > NQ)
02271         ERRMSG("Too many quantities!");
02272     for (int iq = 0; iq < ctl->nq; iq++) {
02273
02274         /* Read quantity name and format... */
02275         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02276         scan_ctl(filename, argc, argv, "QNT_LONGNAME", iq, ctl->qnt_name[iq],
02277             ctl->qnt_longname[iq]);
02278         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02279             ctl->qnt_format[iq]);
02280
02281         /* Try to identify quantity... */
02282         SET_QNT(qnt_idx, "idx", "particle index", "-")
02283         SET_QNT(qnt_ens, "ens", "ensemble index", "-")
02284         SET_QNT(qnt_stat, "stat", "station flag", "-")

```

```

02285     SET_QNT(qnt_m, "m", "mass", "kg")
02286     SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
02287     SET_QNT(qnt_rp, "rp", "particle radius", "microns")
02288     SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
02289     SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
02290     SET_QNT(qnt_ts, "ts", "surface temperature", "K")
02291     SET_QNT(qnt_zs, "zs", "surface height", "km")
02292     SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
02293     SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
02294     SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
02295     SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
02296     SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
02297     SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
02298     SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
02299     SET_QNT(qnt_z, "z", "geopotential height", "km")
02300     SET_QNT(qnt_p, "p", "pressure", "hPa")
02301     SET_QNT(qnt_t, "t", "temperature", "K")
02302     SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
02303     SET_QNT(qnt_u, "u", "zonal wind", "m/s")
02304     SET_QNT(qnt_v, "v", "meridional wind", "m/s")
02305     SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
02306     SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
02307     SET_QNT(qnt_o3, "o3", "ozone", "ppv")
02308     SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
02309     SET_QNT(qnt_iwc, "iwc", "cloud liquid water content", "kg/kg")
02310     SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
02311     SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
02312     SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
02313     SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
02314     SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
02315     SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
02316     SET_QNT(qnt_cape, "cape", "convective available potential energy",
02317         "J/kg")
02318     SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
02319     SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
02320     SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
02321     SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
02322     SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
02323     SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
02324         "kg")
02325     SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
02326         "kg")
02327     SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
02328         "kg")
02329     SET_QNT(qnt_mloss_decay, "mloss_decay",
02330         "mass loss due to exponential decay", "kg")
02331     SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
02332     SET_QNT(qnt_pspace, "pspace", "saturation pressure over ice", "hPa")
02333     SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
02334     SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
02335     SET_QNT(qnt_rh, "rh", "relative humidity", "%")
02336     SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%")
02337     SET_QNT(qnt_theta, "theta", "potential temperature", "K")
02338     SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
02339     SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
02340     SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
02341     SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
02342     SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
02343     SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
02344     SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
02345     SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
02346     SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
02347     SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
02348     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02349 }
02350
02351 /* netCDF I/O parameters... */
02352 ctl->chunkszhint =
02353     (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02354         NULL);
02355 ctl->read_mode =
02356     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02357
02358 /* Vertical coordinates and velocities... */
02359 ctl->vert_coord_ap =
02360     (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
02361 ctl->vert_coord_met =
02362     (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
02363 ctl->vert_vel =
02364     (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
02365 ctl->clams_met_data =
02366     (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);
02367
02368 /* Time steps of simulation... */
02369 ctl->direction =
02370     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02371 if (ctl->direction != -1 && ctl->direction != 1)

```



```

02372     ERRMSG("Set DIRECTION to -1 or 1!");
02373     ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02374     ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02375
02376     /* Meteo data... */
02377     scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02378     ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02379     ctl->met_type =
02380         (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
02381     ctl->met_nc_scale =
02382         (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", NULL);
02383     ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02384     ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02385     ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02386     if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02387         ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02388     ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02389     ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02390     ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02391     if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02392         ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02393     ctl->met_detrend =
02394         scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02395     ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02396     if (ctl->met_np > EP)
02397         ERRMSG("Too many levels!");
02398     for (int ip = 0; ip < ctl->met_np; ip++)
02399         ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02400     ctl->met_geopot_sx
02401         = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02402     ctl->met_geopot_sy
02403         = (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02404     ctl->met_relhumi
02405         = (int) scan_ctl(filename, argc, argv, "MET_RELHUM", -1, "0", NULL);
02406     ctl->met_tropo =
02407         (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02408     if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02409         ERRMSG("Set MET_TROPO = 0 ... 5!");
02410     ctl->met_tropo_lapse =
02411         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02412     ctl->met_tropo_nlev =
02413         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02414     ctl->met_tropo_lapse_sep =
02415         scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02416     ctl->met_tropo_nlev_sep =
02417         (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02418             NULL);
02419     ctl->met_tropo_pv =
02420         scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02421     ctl->met_tropo_theta =
02422         scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02423     ctl->met_tropo_spline =
02424         (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02425     ctl->met_cloud =
02426         (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02427     if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02428         ERRMSG("Set MET_CLOUD = 0 ... 3!");
02429     ctl->met_cloud_min =
02430         scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
02431     ctl->met_dt_out =
02432         scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02433     ctl->met_cache =
02434         (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02435
02436     /* Sorting... */
02437     ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);
02438
02439     /* Isosurface parameters... */
02440     ctl->isosurf =
02441         (int) scan_ctl(filename, argc, argv, "ISOSURE", -1, "0", NULL);
02442     scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02443
02444     /* Advection parameters... */
02445     ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
02446     if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
02447         ERRMSG("Set ADVECT to 1, 2, or 4!");
02448     ctl->reflect =
02449         (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02450
02451     /* Diffusion parameters... */
02452     ctl->turb_dx_trop =
02453         scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02454     ctl->turb_dx_strat =
02455         scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02456     ctl->turb_dz_trop =
02457         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02458     ctl->turb_dz_strat =

```

```

02459     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02460     ctl->turb_mesox =
02461     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02462     ctl->turb_mesoz =
02463     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02464
02465     /* Convection... */
02466     ctl->conv_cape
02467     = scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02468     ctl->conv_cin
02469     = scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02470     ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02471     ctl->conv_mix
02472     = (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "1", NULL);
02473     ctl->conv_mix_bot
02474     = (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02475     ctl->conv_mix_top
02476     = (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02477
02478     /* Boundary conditions... */
02479     ctl->bound_mass =
02480     scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02481     ctl->bound_mass_trend =
02482     scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
02483     ctl->bound_vmr =
02484     scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02485     ctl->bound_vmr_trend =
02486     scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
02487     ctl->bound_lat0 =
02488     scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02489     ctl->bound_lat1 =
02490     scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02491     ctl->bound_p0 =
02492     scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02493     ctl->bound_p1 =
02494     scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02495     ctl->bound_dps =
02496     scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02497     ctl->bound_dzs =
02498     scan_ctl(filename, argc, argv, "BOUND_DZS", -1, "-999", NULL);
02499     ctl->bound_zetas =
02500     scan_ctl(filename, argc, argv, "BOUND_ZETAS", -1, "-999", NULL);
02501     ctl->bound_pbl =
02502     (int) scan_ctl(filename, argc, argv, "BOUND_PBL", -1, "0", NULL);
02503
02504     /* Species parameters... */
02505     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02506     if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02507         ctl->molmass = 120.907;
02508         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
02509         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
02510     } else if (strcasecmp(ctl->species, "CFCl3") == 0) {
02511         ctl->molmass = 137.359;
02512         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
02513         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
02514     } else if (strcasecmp(ctl->species, "CH4") == 0) {
02515         ctl->molmass = 16.043;
02516         ctl->oh_chem_reaction = 2;
02517         ctl->oh_chem[0] = 2.45e-12;
02518         ctl->oh_chem[1] = 1775;
02519         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
02520         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02521     } else if (strcasecmp(ctl->species, "CO") == 0) {
02522         ctl->molmass = 28.01;
02523         ctl->oh_chem_reaction = 3;
02524         ctl->oh_chem[0] = 6.9e-33;
02525         ctl->oh_chem[1] = 2.1;
02526         ctl->oh_chem[2] = 1.1e-12;
02527         ctl->oh_chem[3] = -1.3;
02528         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
02529         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;
02530     } else if (strcasecmp(ctl->species, "CO2") == 0) {
02531         ctl->molmass = 44.009;
02532         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
02533         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02534     } else if (strcasecmp(ctl->species, "H2O") == 0) {
02535         ctl->molmass = 18.01528;
02536     } else if (strcasecmp(ctl->species, "N2O") == 0) {
02537         ctl->molmass = 44.013;
02538         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
02539         ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
02540     } else if (strcasecmp(ctl->species, "NH3") == 0) {
02541         ctl->molmass = 17.031;
02542         ctl->oh_chem_reaction = 2;
02543         ctl->oh_chem[0] = 1.7e-12;
02544         ctl->oh_chem[1] = 710;
02545         ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;

```

```
02546     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
02547 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02548     ctl->molmass = 63.012;
02549     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
02550     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
02551 } else if (strcasecmp(ctl->species, "NO") == 0) {
02552     ctl->molmass = 30.006;
02553     ctl->oh_chem_reaction = 3;
02554     ctl->oh_chem[0] = 7.1e-31;
02555     ctl->oh_chem[1] = 2.6;
02556     ctl->oh_chem[2] = 3.6e-11;
02557     ctl->oh_chem[3] = 0.1;
02558     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
02559     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02560 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02561     ctl->molmass = 46.005;
02562     ctl->oh_chem_reaction = 3;
02563     ctl->oh_chem[0] = 1.8e-30;
02564     ctl->oh_chem[1] = 3.0;
02565     ctl->oh_chem[2] = 2.8e-11;
02566     ctl->oh_chem[3] = 0.0;
02567     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
02568     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02569 } else if (strcasecmp(ctl->species, "O3") == 0) {
02570     ctl->molmass = 47.997;
02571     ctl->oh_chem_reaction = 2;
02572     ctl->oh_chem[0] = 1.7e-12;
02573     ctl->oh_chem[1] = 940;
02574     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
02575     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
02576 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02577     ctl->molmass = 146.048;
02578     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
02579     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
02580 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02581     ctl->molmass = 64.066;
02582     ctl->oh_chem_reaction = 3;
02583     ctl->oh_chem[0] = 2.9e-31;
02584     ctl->oh_chem[1] = 4.1;
02585     ctl->oh_chem[2] = 1.7e-12;
02586     ctl->oh_chem[3] = -0.2;
02587     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
02588     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
02589 } else {
02590     ctl->molmass =
02591         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02592     ctl->oh_chem_reaction =
02593         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02594     ctl->h2o2_chem_reaction =
02595         (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
02596             NULL);
02597     for (int ip = 0; ip < 4; ip++)
02598         ctl->oh_chem[ip] =
02599             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02600     ctl->dry_depo_vdep =
02601         scan_ctl(filename, argc, argv, "DRY_DEPO_VDEP", -1, "0", NULL);
02602     ctl->dry_depo_dp =
02603         scan_ctl(filename, argc, argv, "DRY_DEPO_DP", -1, "30", NULL);
02604     ctl->wet_depo_ic_a =
02605         scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
02606     ctl->wet_depo_ic_b =
02607         scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
02608     ctl->wet_depo_bc_a =
02609         scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
02610     ctl->wet_depo_bc_b =
02611         scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
02612     for (int ip = 0; ip < 3; ip++)
02613         ctl->wet_depo_ic_h[ip] =
02614             scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
02615     for (int ip = 0; ip < 1; ip++)
02616         ctl->wet_depo_bc_h[ip] =
02617             scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
02618 }
02619
02620 /* Wet deposition... */
02621 ctl->wet_depo_pre[0] =
02622     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
02623 ctl->wet_depo_pre[1] =
02624     scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
02625 ctl->wet_depo_ic_ret_ratio =
02626     scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
02627 ctl->wet_depo_bc_ret_ratio =
02628     scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);
02629
02630 /* OH chemistry... */
02631 ctl->oh_chem_beta =
02632     scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
```

```

02633 scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
02634             "../data/clams_radical_species.nc", ctl->clim_oh_filename);
02635
02636 /* H2O2 chemistry... */
02637 ctl->h2o2_chem_cc =
02638     scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
02639 scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
02640             "../data/cams_H2O2.nc", ctl->clim_h2o2_filename);
02641
02642 /* Chemistry grid... */
02643 ctl->chemgrid_z0 =
02644     scan_ctl(filename, argc, argv, "CHEMGRID_Z0", -1, "0", NULL);
02645 ctl->chemgrid_z1 =
02646     scan_ctl(filename, argc, argv, "CHEMGRID_Z1", -1, "100", NULL);
02647 ctl->chemgrid_nz =
02648     (int) scan_ctl(filename, argc, argv, "CHEMGRID_NZ", -1, "1", NULL);
02649 ctl->chemgrid_lon0 =
02650     scan_ctl(filename, argc, argv, "CHEMGRID_LON0", -1, "-180", NULL);
02651 ctl->chemgrid_lon1 =
02652     scan_ctl(filename, argc, argv, "CHEMGRID_LON1", -1, "180", NULL);
02653 ctl->chemgrid_nx =
02654     (int) scan_ctl(filename, argc, argv, "CHEMGRID_NX", -1, "360", NULL);
02655 ctl->chemgrid_lat0 =
02656     scan_ctl(filename, argc, argv, "CHEMGRID_LAT0", -1, "-90", NULL);
02657 ctl->chemgrid_lat1 =
02658     scan_ctl(filename, argc, argv, "CHEMGRID_LAT1", -1, "90", NULL);
02659 ctl->chemgrid_ny =
02660     (int) scan_ctl(filename, argc, argv, "CHEMGRID_NY", -1, "180", NULL);
02661
02662 /* Exponential decay... */
02663 ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02664 ctl->tdec_strat =
02665     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02666
02667 /* PSC analysis... */
02668 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02669 ctl->psc_hno3 =
02670     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02671
02672 /* Output of atmospheric data... */
02673 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02674 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02675 ctl->atm_dt_out =
02676     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02677 ctl->atm_filter =
02678     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02679 ctl->atm_stride =
02680     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02681 ctl->atm_type =
02682     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02683
02684 /* Output of CSI data... */
02685 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02686 ctl->csi_dt_out =
02687     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02688 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02689 ctl->csi_obsmin =
02690     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02691 ctl->csi_modmin =
02692     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02693 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02694 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02695 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02696 ctl->csi_lon0 =
02697     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02698 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02699 ctl->csi_nx =
02700     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02701 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02702 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02703 ctl->csi_ny =
02704     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02705
02706 /* Output of ensemble data... */
02707 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02708 ctl->ens_dt_out =
02709     scan_ctl(filename, argc, argv, "ENS_DT_OUT", -1, "86400", NULL);
02710
02711 /* Output of grid data... */
02712 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02713             ctl->grid_basename);
02714 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02715 ctl->grid_dt_out =
02716     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02717 ctl->grid_sparse =
02718     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02719 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);

```

```

02720     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02721     ctl->grid_nz =
02722         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02723     ctl->grid_lon0 =
02724         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02725     ctl->grid_lon1 =
02726         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02727     ctl->grid_nx =
02728         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02729     ctl->grid_lat0 =
02730         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02731     ctl->grid_lat1 =
02732         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02733     ctl->grid_ny =
02734         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02735     ctl->grid_type =
02736         (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", NULL);
02737
02738     /* Output of profile data... */
02739     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02740         ctl->prof_basename);
02741     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02742     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02743     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02744     ctl->prof_nz =
02745         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02746     ctl->prof_lon0 =
02747         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02748     ctl->prof_lon1 =
02749         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02750     ctl->prof_nx =
02751         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02752     ctl->prof_lat0 =
02753         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02754     ctl->prof_lat1 =
02755         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02756     ctl->prof_ny =
02757         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02758
02759     /* Output of sample data... */
02760     scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02761         ctl->sample_basename);
02762     scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02763         ctl->sample_obsfile);
02764     ctl->sample_dx =
02765         scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02766     ctl->sample_dz =
02767         scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02768
02769     /* Output of station data... */
02770     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02771         ctl->stat_basename);
02772     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02773     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02774     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02775     ctl->stat_t0 =
02776         scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02777     ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02778 }
02779
02780 /*****
02781
02782 int read_met(
02783     char *filename,
02784     ctl_t * ctl,
02785     clim_t * clim,
02786     met_t * met) {
02787
02788     /* Write info... */
02789     LOG(1, "Read meteo data: %s", filename);
02790
02791     /* Read netCDF data... */
02792     if (ctl->met_type == 0) {
02793
02794         int ncid;
02795
02796         /* Open netCDF file... */
02797         if (nc__open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02798             NC_NOERR) {
02799             WARN("Cannot open file!");
02800             return 0;
02801         }
02802
02803         /* Read coordinates of meteo data... */
02804         read_met_grid(filename, ncid, ctl, met);
02805
02806         /* Read meteo data on vertical levels... */

```

```

02807     read_met_levels(ncid, ctl, met);
02808
02809     /* Extrapolate data for lower boundary... */
02810     read_met_extrapolate(met);
02811
02812     /* Read surface data... */
02813     read_met_surface(ncid, met, ctl);
02814
02815     /* Create periodic boundary conditions... */
02816     read_met_periodic(met);
02817
02818     /* Downsampling... */
02819     read_met_sample(ctl, met);
02820
02821     /* Calculate geopotential heights... */
02822     read_met_geopot(ctl, met);
02823
02824     /* Calculate potential vorticity... */
02825     read_met_pv(met);
02826
02827     /* Calculate boundary layer data... */
02828     read_met_pbl(met);
02829
02830     /* Calculate tropopause data... */
02831     read_met_tropo(ctl, clim, met);
02832
02833     /* Calculate cloud properties... */
02834     read_met_cloud(ctl, met);
02835
02836     /* Calculate convective available potential energy... */
02837     read_metCAPE(clim, met);
02838
02839     /* Detrending... */
02840     read_met_detrend(ctl, met);
02841
02842     /* Close file... */
02843     NC(nc_close(ncid));
02844 }
02845
02846 /* Read binary data... */
02847 else if (ctl->met_type >= 1 && ctl->met_type <= 4) {
02848
02849     FILE *in;
02850
02851     double r;
02852
02853     int year, mon, day, hour, min, sec;
02854
02855     /* Set timer... */
02856     SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);
02857
02858     /* Open file... */
02859     if (!(in = fopen(filename, "r"))) {
02860         WARN("Cannot open file!");
02861         return 0;
02862     }
02863
02864     /* Check type of binary data... */
02865     int met_type;
02866     FREAD(&met_type, int,
02867         1,
02868         in);
02869     if (met_type != ctl->met_type)
02870         ERRMSG("Wrong MET_TYPE of binary data!");
02871
02872     /* Check version of binary data... */
02873     int version;
02874     FREAD(&version, int,
02875         1,
02876         in);
02877     if (version != 100)
02878         ERRMSG("Wrong version of binary data!");
02879
02880     /* Read time... */
02881     FREAD(&met->time, double,
02882         1,
02883         in);
02884     jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
02885     LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
02886         met->time, year, mon, day, hour, min);
02887     if (year < 1900 || year > 2100 || mon < 1 || mon > 12
02888         || day < 1 || day > 31 || hour < 0 || hour > 23)
02889         ERRMSG("Error while reading time!");
02890
02891     /* Read dimensions... */
02892     FREAD(&met->nx, int,
02893         1,

```

```

02894         in);
02895     LOG(2, "Number of longitudes: %d", met->nx);
02896     if (met->nx < 2 || met->nx > EX)
02897         ERRMSG("Number of longitudes out of range!");
02898
02899     FREAD(&met->ny, int,
02900          1,
02901          in);
02902     LOG(2, "Number of latitudes: %d", met->ny);
02903     if (met->ny < 2 || met->ny > EY)
02904         ERRMSG("Number of latitudes out of range!");
02905
02906     FREAD(&met->np, int,
02907          1,
02908          in);
02909     LOG(2, "Number of levels: %d", met->np);
02910     if (met->np < 2 || met->np > EP)
02911         ERRMSG("Number of levels out of range!");
02912
02913     /* Read grid... */
02914     FREAD(met->lon, double,
02915          (size_t) met->nx,
02916          in);
02917     LOG(2, "Longitudes: %g, %g ... %g deg",
02918          met->lon[0], met->lon[1], met->lon[met->nx - 1]);
02919
02920     FREAD(met->lat, double,
02921          (size_t) met->ny,
02922          in);
02923     LOG(2, "Latitudes: %g, %g ... %g deg",
02924          met->lat[0], met->lat[1], met->lat[met->ny - 1]);
02925
02926     FREAD(met->p, double,
02927          (size_t) met->np,
02928          in);
02929     LOG(2, "Altitude levels: %g, %g ... %g km",
02930          Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
02931     LOG(2, "Pressure levels: %g, %g ... %g hPa",
02932          met->p[0], met->p[1], met->p[met->np - 1]);
02933
02934     /* Read surface data... */
02935     read_met_bin_2d(in, met, met->ps, "PS");
02936     read_met_bin_2d(in, met, met->ts, "TS");
02937     read_met_bin_2d(in, met, met->zs, "ZS");
02938     read_met_bin_2d(in, met, met->us, "US");
02939     read_met_bin_2d(in, met, met->vs, "VS");
02940     read_met_bin_2d(in, met, met->pbl, "PBL");
02941     read_met_bin_2d(in, met, met->pt, "PT");
02942     read_met_bin_2d(in, met, met->tt, "TT");
02943     read_met_bin_2d(in, met, met->zt, "ZT");
02944     read_met_bin_2d(in, met, met->h2ot, "H2OT");
02945     read_met_bin_2d(in, met, met->pct, "PCT");
02946     read_met_bin_2d(in, met, met->pcb, "PCB");
02947     read_met_bin_2d(in, met, met->cl, "CL");
02948     read_met_bin_2d(in, met, met->plcl, "PLCL");
02949     read_met_bin_2d(in, met, met->plfc, "PLFC");
02950     read_met_bin_2d(in, met, met->pel, "PEL");
02951     read_met_bin_2d(in, met, met->cape, "CAPE");
02952     read_met_bin_2d(in, met, met->cin, "CIN");
02953
02954     /* Read level data... */
02955     read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
02956     read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
02957     read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
02958     read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
02959     read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
02960     read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
02961     read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
02962     read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
02963     read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
02964     read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);
02965
02966     /* Read final flag... */
02967     int final;
02968     FREAD(&final, int,
02969          1,
02970          in);
02971     if (final != 999)
02972         ERRMSG("Error while reading binary data!");
02973
02974     /* Close file... */
02975     fclose(in);
02976 }
02977
02978 /* Not implemented... */
02979 else
02980     ERRMSG("MET_TYPE not implemented!");

```

```

02981
02982 /* Copy wind data to cache... */
02983 #ifndef UVW
02984 #pragma omp parallel for default(shared) collapse(2)
02985 for (int ix = 0; ix < met->nx; ix++)
02986     for (int iy = 0; iy < met->ny; iy++)
02987         for (int ip = 0; ip < met->np; ip++) {
02988             met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
02989             met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
02990             met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
02991         }
02992 #endif
02993
02994 /* Return success... */
02995 return 1;
02996 }
02997
02998 /*****
02999
03000 void read_met_bin_2d(
03001     FILE * in,
03002     met_t * met,
03003     float var[EX][EY],
03004     char *varname) {
03005
03006     float *help;
03007
03008     /* Allocate... */
03009     ALLOC(help, float,
03010          EX * EY);
03011
03012     /* Read uncompressed... */
03013     LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
03014     FREAD(help, float,
03015           (size_t) (met->nx * met->ny),
03016           in);
03017
03018     /* Copy data... */
03019     for (int ix = 0; ix < met->nx; ix++)
03020         for (int iy = 0; iy < met->ny; iy++)
03021             var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];
03022
03023     /* Free... */
03024     free(help);
03025 }
03026
03027 /*****
03028
03029 void read_met_bin_3d(
03030     FILE * in,
03031     ctl_t * ctl,
03032     met_t * met,
03033     float var[EX][EY][EP],
03034     char *varname,
03035     int precision,
03036     double tolerance) {
03037
03038     float *help;
03039
03040     /* Allocate... */
03041     ALLOC(help, float,
03042          EX * EY * EP);
03043
03044     /* Read uncompressed data... */
03045     if (ctl->met_type == 1) {
03046         LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
03047         FREAD(help, float,
03048             (size_t) (met->nx * met->ny * met->np),
03049             in);
03050     }
03051
03052     /* Read packed data... */
03053     else if (ctl->met_type == 2)
03054         compress_pack(varname, help, (size_t) (met->ny * met->nx),
03055                      (size_t) met->np, 1, in);
03056
03057     /* Read zfp data... */
03058     else if (ctl->met_type == 3) {
03059 #ifndef ZFP
03060         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
03061                     tolerance, 1, in);
03062 #else
03063         ERRMSG("zfp compression not supported!");
03064         LOG(3, "%d %g", precision, tolerance);
03065 #endif
03066     }
03067

```



```

03068  /* Read zstd data... */
03069  else if (ctl->met_type == 4) {
03070  #ifdef ZSTD
03071      compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
03072                  in);
03073  #else
03074      ERRMSG("zstd compression not supported!");
03075  #endif
03076  }
03077
03078  /* Copy data... */
03079  #pragma omp parallel for default(shared) collapse(2)
03080  for (int ix = 0; ix < met->nx; ix++)
03081      for (int iy = 0; iy < met->ny; iy++)
03082          for (int ip = 0; ip < met->np; ip++)
03083              var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];
03084
03085  /* Free... */
03086  free(help);
03087 }
03088
03089 /*****
03090 void read_met_cape(
03091     clim_t * clim,
03092     met_t * met) {
03093
03094
03095  /* Set timer... */
03096  SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
03097  LOG(2, "Calculate CAPE...");
03098
03099  /* Vertical spacing (about 100 m)... */
03100  const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
03101
03102  /* Loop over columns... */
03103  #pragma omp parallel for default(shared) collapse(2)
03104  for (int ix = 0; ix < met->nx; ix++)
03105      for (int iy = 0; iy < met->ny; iy++) {
03106
03107          /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
03108          int n = 0;
03109          double h2o = 0, t, theta = 0;
03110          double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
03111          double ptop = pbot - 50.;
03112          for (int ip = 0; ip < met->np; ip++) {
03113              if (met->p[ip] <= pbot) {
03114                  theta += THETA(met->p[ip], met->t[ix][iy][ip]);
03115                  h2o += met->h2o[ix][iy][ip];
03116                  n++;
03117              }
03118              if (met->p[ip] < ptop && n > 0)
03119                  break;
03120          }
03121          theta /= n;
03122          h2o /= n;
03123
03124          /* Cannot compute anything if water vapor is missing... */
03125          met->plcl[ix][iy] = GSL_NAN;
03126          met->plfc[ix][iy] = GSL_NAN;
03127          met->pel[ix][iy] = GSL_NAN;
03128          met->cape[ix][iy] = GSL_NAN;
03129          met->cin[ix][iy] = GSL_NAN;
03130          if (h2o <= 0)
03131              continue;
03132
03133          /* Find lifted condensation level (LCL)... */
03134          ptop = P(20.);
03135          pbot = met->ps[ix][iy];
03136          do {
03137              met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
03138              t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
03139              if (RH(met->plcl[ix][iy], t, h2o) > 100.)
03140                  ptop = met->plcl[ix][iy];
03141              else
03142                  pbot = met->plcl[ix][iy];
03143          } while (pbot - ptop > 0.1);
03144
03145          /* Calculate CIN up to LCL... */
03146          INTPOL_INIT;
03147          double dcaps, dz, h2o_env, t_env;
03148          double p = met->ps[ix][iy];
03149          met->cape[ix][iy] = met->cin[ix][iy] = 0;
03150          do {
03151              dz = dz0 * TVIRT(t, h2o);
03152              p /= pfac;
03153              t = theta / pow(1000. / p, 0.286);
03154              intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],

```

```

03155         &t_env, ci, cw, 1);
03156     intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03157         &h2o_env, ci, cw, 0);
03158     dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03159         TVIRT(t_env, h2o_env) * dz;
03160     if (dcape < 0)
03161         met->cin[ix][iy] += fabsf((float) dcape);
03162     } while (p > met->plcl[ix][iy]);
03163
03164     /* Calculate level of free convection (LFC), equilibrium level (EL),
03165        and convective available potential energy (CAPE)... */
03166     dcape = 0;
03167     p = met->plcl[ix][iy];
03168     t = theta / pow(1000. / p, 0.286);
03169     ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
03170     do {
03171         dz = dz0 * TVIRT(t, h2o);
03172         p /= pfac;
03173         t -= lapse_rate(t, h2o) * dz;
03174         double psat = PSAT(t);
03175         h2o = psat / (p - (1. - EPS) * psat);
03176         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03177             &t_env, ci, cw, 1);
03178         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03179             &h2o_env, ci, cw, 0);
03180         double dcape_old = dcape;
03181         dcape = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03182             TVIRT(t_env, h2o_env) * dz;
03183         if (dcape > 0) {
03184             met->cape[ix][iy] += (float) dcape;
03185             if (!isfinite(met->plfc[ix][iy]))
03186                 met->plfc[ix][iy] = (float) p;
03187         } else if (dcape_old > 0)
03188             met->pel[ix][iy] = (float) p;
03189         if (dcape < 0 && !isfinite(met->plfc[ix][iy]))
03190             met->cin[ix][iy] += fabsf((float) dcape);
03191     } while (p > ptop);
03192
03193     /* Check results... */
03194     if (!isfinite(met->plfc[ix][iy]))
03195         met->cin[ix][iy] = GSL_NAN;
03196 }
03197 }
03198
03199 /*****
03200
03201 void read_met_cloud(
03202     ctl_t * ctl,
03203     met_t * met) {
03204
03205     /* Set timer... */
03206     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
03207     LOG(2, "Calculate cloud data...");
03208
03209     /* Loop over columns... */
03210     #pragma omp parallel for default(shared) collapse(2)
03211     for (int ix = 0; ix < met->nx; ix++)
03212         for (int iy = 0; iy < met->ny; iy++) {
03213
03214             /* Init... */
03215             met->pct[ix][iy] = GSL_NAN;
03216             met->pcb[ix][iy] = GSL_NAN;
03217             met->cl[ix][iy] = 0;
03218
03219             /* Loop over pressure levels... */
03220             for (int ip = 0; ip < met->np - 1; ip++) {
03221
03222                 /* Check pressure... */
03223                 if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
03224                     continue;
03225
03226                 /* Check ice water and liquid water content... */
03227                 if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
03228                     || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {
03229
03230                     /* Get cloud top pressure ... */
03231                     met->pct[ix][iy]
03232                         = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
03233
03234                     /* Get cloud bottom pressure ... */
03235                     if (!isfinite(met->pcb[ix][iy]))
03236                         met->pcb[ix][iy]
03237                             = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
03238                 }
03239
03240                 /* Get cloud water... */
03241                 met->cl[ix][iy] += (float)

```

```

03242         (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
03243             + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
03244             * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
03245     }
03246 }
03247 }
03248
03249 /*****
03250
03251 void read_met_detrend(
03252     ctl_t * ctl,
03253     met_t * met) {
03254
03255     met_t *help;
03256
03257     /* Check parameters... */
03258     if (ctl->met_detrend <= 0)
03259         return;
03260
03261     /* Set timer... */
03262     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
03263     LOG(2, "Detrend meteo data...");
03264
03265     /* Allocate... */
03266     ALLOC(help, met_t, 1);
03267
03268     /* Calculate standard deviation... */
03269     double sigma = ctl->met_detrend / 2.355;
03270     double tssq = 2. * SQR(sigma);
03271
03272     /* Calculate box size in latitude... */
03273     int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03274     sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03275
03276     /* Calculate background... */
03277     #pragma omp parallel for default(shared) collapse(2)
03278     for (int ix = 0; ix < met->nx; ix++) {
03279         for (int iy = 0; iy < met->ny; iy++) {
03280
03281             /* Calculate Cartesian coordinates... */
03282             double x0[3];
03283             geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03284
03285             /* Calculate box size in longitude... */
03286             int sx =
03287                 (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03288                     fabs(met->lon[1] - met->lon[0]));
03289             sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03290
03291             /* Init... */
03292             float wsum = 0;
03293             for (int ip = 0; ip < met->np; ip++) {
03294                 help->t[ix][iy][ip] = 0;
03295                 help->u[ix][iy][ip] = 0;
03296                 help->v[ix][iy][ip] = 0;
03297                 help->w[ix][iy][ip] = 0;
03298             }
03299
03300             /* Loop over neighboring grid points... */
03301             for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03302                 int ix3 = ix2;
03303                 if (ix3 < 0)
03304                     ix3 += met->nx;
03305                 else if (ix3 >= met->nx)
03306                     ix3 -= met->nx;
03307                 for (int iy2 = GSL_MAX(iy - sy, 0);
03308                     iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03309
03310                     /* Calculate Cartesian coordinates... */
03311                     double x1[3];
03312                     geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03313
03314                     /* Calculate weighting factor... */
03315                     float w = (float) exp(-DIST2(x0, x1) / tssq);
03316
03317                     /* Add data... */
03318                     wsum += w;
03319                     for (int ip = 0; ip < met->np; ip++) {
03320                         help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03321                         help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03322                         help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03323                         help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03324                     }
03325                 }
03326             }
03327
03328             /* Normalize... */

```

```

03329     for (int ip = 0; ip < met->np; ip++) {
03330         help->t[ix][iy][ip] /= wsum;
03331         help->u[ix][iy][ip] /= wsum;
03332         help->v[ix][iy][ip] /= wsum;
03333         help->w[ix][iy][ip] /= wsum;
03334     }
03335 }
03336 }
03337
03338 /* Subtract background... */
03339 #pragma omp parallel for default(shared) collapse(3)
03340 for (int ix = 0; ix < met->nx; ix++)
03341     for (int iy = 0; iy < met->ny; iy++)
03342         for (int ip = 0; ip < met->np; ip++) {
03343             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03344             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03345             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03346             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03347         }
03348
03349 /* Free... */
03350 free(help);
03351 }
03352
03353 /*****
03354 void read_met_extrapolate(
03355     met_t * met) {
03356
03357     /* Set timer... */
03358     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03359     LOG(2, "Extrapolate meteo data...");
03360
03361     /* Loop over columns... */
03362     #pragma omp parallel for default(shared) collapse(2)
03363     for (int ix = 0; ix < met->nx; ix++)
03364         for (int iy = 0; iy < met->ny; iy++) {
03365
03366             /* Find lowest valid data point... */
03367             int ip0;
03368             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03369                 if (!isfinite(met->t[ix][iy][ip0])
03370                     || !isfinite(met->u[ix][iy][ip0])
03371                     || !isfinite(met->v[ix][iy][ip0])
03372                     || !isfinite(met->w[ix][iy][ip0]))
03373                     break;
03374
03375             /* Extrapolate... */
03376             for (int ip = ip0; ip >= 0; ip--) {
03377                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03378                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03379                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03380                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03381                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03382                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03383                 met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03384                 met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03385             }
03386         }
03387     }
03388 }
03389
03390 /*****
03391 void read_met_geopot(
03392     ctl_t * ctl,
03393     met_t * met) {
03394
03395     static float help[EP][EX][EY];
03396
03397     double logp[EP];
03398
03399     int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03400
03401     /* Set timer... */
03402     SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03403     LOG(2, "Calculate geopotential heights...");
03404
03405     /* Calculate log pressure... */
03406     #pragma omp parallel for default(shared)
03407     for (int ip = 0; ip < met->np; ip++)
03408         logp[ip] = log(met->p[ip]);
03409
03410     /* Apply hydrostatic equation to calculate geopotential heights... */
03411     #pragma omp parallel for default(shared) collapse(2)
03412     for (int ix = 0; ix < met->nx; ix++)
03413         for (int iy = 0; iy < met->ny; iy++) {
03414
03415

```

```

03416     /* Get surface height and pressure... */
03417     double zs = met->zs[ix][iy];
03418     double lnps = log(met->ps[ix][iy]);
03419
03420     /* Get temperature and water vapor vmr at the surface... */
03421     int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03422     double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03423                     met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03424     double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03425                       met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03426
03427     /* Upper part of profile... */
03428     met->z[ix][iy][ip0 + 1]
03429     = (float) (zs +
03430               ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03431                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03432     for (int ip = ip0 + 2; ip < met->np; ip++)
03433         met->z[ix][iy][ip]
03434         = (float) (met->z[ix][iy][ip - 1] +
03435                   ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03436                         met->h2o[ix][iy][ip - 1], logp[ip],
03437                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03438
03439     /* Lower part of profile... */
03440     met->z[ix][iy][ip0]
03441     = (float) (zs +
03442               ZDIFF(lnps, ts, h2os, logp[ip0],
03443                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03444     for (int ip = ip0 - 1; ip >= 0; ip--)
03445         met->z[ix][iy][ip]
03446         = (float) (met->z[ix][iy][ip + 1] +
03447                   ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03448                         met->h2o[ix][iy][ip + 1], logp[ip],
03449                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03450 }
03451
03452 /* Check control parameters... */
03453 if (dx == 0 || dy == 0)
03454     return;
03455
03456 /* Default smoothing parameters... */
03457 if (dx < 0 || dy < 0) {
03458     if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03459         dx = 3;
03460         dy = 2;
03461     } else {
03462         dx = 6;
03463         dy = 4;
03464     }
03465 }
03466
03467 /* Calculate weights for smoothing... */
03468 float ws[dx + 1][dy + 1];
03469 #pragma omp parallel for default(shared) collapse(2)
03470 for (int ix = 0; ix <= dx; ix++)
03471     for (int iy = 0; iy < dy; iy++)
03472         ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03473             * (1.0f - (float) iy / (float) dy);
03474
03475 /* Copy data... */
03476 #pragma omp parallel for default(shared) collapse(3)
03477 for (int ix = 0; ix < met->nx; ix++)
03478     for (int iy = 0; iy < met->ny; iy++)
03479         for (int ip = 0; ip < met->np; ip++)
03480             help[ip][ix][iy] = met->z[ix][iy][ip];
03481
03482 /* Horizontal smoothing... */
03483 #pragma omp parallel for default(shared) collapse(3)
03484 for (int ip = 0; ip < met->np; ip++)
03485     for (int ix = 0; ix < met->nx; ix++)
03486         for (int iy = 0; iy < met->ny; iy++) {
03487             float res = 0, wsum = 0;
03488             int iy0 = GSL_MAX(iy - dy + 1, 0);
03489             int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03490             for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03491                 int ix3 = ix2;
03492                 if (ix3 < 0)
03493                     ix3 += met->nx;
03494                 else if (ix3 >= met->nx)
03495                     ix3 -= met->nx;
03496                 for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03497                     if (isfinite(help[ip][ix3][iy2])) {
03498                         float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03499                         res += w * help[ip][ix3][iy2];
03500                         wsum += w;
03501                     }
03502             }

```

```

03503         if (wsum > 0)
03504             met->z[ix][iy][ip] = res / wsum;
03505         else
03506             met->z[ix][iy][ip] = GSL_NAN;
03507     }
03508 }
03509
03510 /*****
03511
03512 void read_met_grid(
03513     char *filename,
03514     int ncid,
03515     ctl_t * ctl,
03516     met_t * met) {
03517
03518     char levname[LEN], tstr[10];
03519
03520     double rtime, r2;
03521
03522     int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;
03523
03524     size_t np;
03525
03526     /* Set timer... */
03527     SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03528     LOG(2, "Read meteo grid information...");
03529
03530     /* MPTRAC meteo files... */
03531     if (ctl->clams_met_data == 0) {
03532
03533         /* Get time from filename... */
03534         size_t len = strlen(filename);
03535         sprintf(tstr, "%.4s", &filename[len - 16]);
03536         year = atoi(tstr);
03537         sprintf(tstr, "%.2s", &filename[len - 11]);
03538         mon = atoi(tstr);
03539         sprintf(tstr, "%.2s", &filename[len - 8]);
03540         day = atoi(tstr);
03541         sprintf(tstr, "%.2s", &filename[len - 5]);
03542         hour = atoi(tstr);
03543         time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03544
03545         /* Check time information from data file... */
03546         if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03547             NC(nc_get_var_double(ncid, varid, &rtime));
03548             if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rtime) > 1.0)
03549                 WARN("Time information in meteo file does not match filename!");
03550             } else
03551                 WARN("Time information in meteo file is missing!");
03552         }
03553
03554         /* CLaMS meteo files... */
03555         else {
03556
03557             /* Read time from file... */
03558             NC_GET_DOUBLE("time", &rtime, 0);
03559
03560             /* Get time from filename (considering the century)... */
03561             if (rtime < 0)
03562                 sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
03563             else
03564                 sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
03565             year = atoi(tstr);
03566             sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
03567             mon = atoi(tstr);
03568             sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
03569             day = atoi(tstr);
03570             sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03571             hour = atoi(tstr);
03572             time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03573         }
03574
03575         /* Check time... */
03576         if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03577             || day < 1 || day > 31 || hour < 0 || hour > 23)
03578             ERRMSG("Cannot read time from filename!");
03579         jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03580         LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03581             met->time, year2, mon2, day2, hour2, min2);
03582
03583         /* Get grid dimensions... */
03584         NC_INQ_DIM("lon", &met->nx, 2, EX);
03585         LOG(2, "Number of longitudes: %d", met->nx);
03586
03587         NC_INQ_DIM("lat", &met->ny, 2, EY);
03588         LOG(2, "Number of latitudes: %d", met->ny);
03589

```

```

03590     if (ctl->vert_coord_met == 0) {
03591         int dimid;
03592         sprintf(levname, "lev");
03593         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR)
03594             sprintf(levname, "plev");
03595     } else
03596         sprintf(levname, "hybrid");
03597     NC_INQ_DIM(levname, &met->np, 1, EP);
03598     if (met->np == 1) {
03599         int dimid;
03600         sprintf(levname, "lev_2");
03601         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03602             sprintf(levname, "plev");
03603             nc_inq_dimid(ncid, levname, &dimid);
03604         }
03605         NC(nc_inq_dimlen(ncid, dimid, &np));
03606         met->np = (int) np;
03607     }
03608     LOG(2, "Number of levels: %d", met->np);
03609     if (met->np < 2 || met->np > EP)
03610         ERRMSG("Number of levels out of range!");
03611
03612     /* Read longitudes and latitudes... */
03613     NC_GET_DOUBLE("lon", met->lon, 1);
03614     LOG(2, "Longitudes: %g, %g ... %g deg",
03615         met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03616     NC_GET_DOUBLE("lat", met->lat, 1);
03617     LOG(2, "Latitudes: %g, %g ... %g deg",
03618         met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03619
03620     /* Read pressure levels... */
03621     if (ctl->met_np <= 0) {
03622         NC_GET_DOUBLE(levname, met->p, 1);
03623         for (int ip = 0; ip < met->np; ip++)
03624             met->p[ip] /= 100.;
03625         LOG(2, "Altitude levels: %g, %g ... %g km",
03626             Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
03627         LOG(2, "Pressure levels: %g, %g ... %g hPa",
03628             met->p[0], met->p[1], met->p[met->np - 1]);
03629     }
03630 }
03631
03632 /*****
03633
03634 void read_met_levels(
03635     int ncid,
03636     ctl_t * ctl,
03637     met_t * met) {
03638
03639     /* Set timer... */
03640     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03641     LOG(2, "Read level data...");
03642
03643     /* MPTRAC meteo data... */
03644     if (ctl->clams_met_data == 0) {
03645
03646         /* Read meteo data... */
03647         if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
03648             ERRMSG("Cannot read temperature!");
03649         if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03650             ERRMSG("Cannot read zonal wind!");
03651         if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03652             ERRMSG("Cannot read meridional wind!");
03653         if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
03654             WARN("Cannot read vertical velocity!");
03655         if (!read_met_nc_3d
03656             (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03657             WARN("Cannot read specific humidity!");
03658         if (!read_met_nc_3d
03659             (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03660             WARN("Cannot read ozone data!");
03661         if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03662             if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03663                 WARN("Cannot read cloud liquid water content!");
03664             if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03665                 WARN("Cannot read cloud ice water content!");
03666         }
03667         if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03668             if (!read_met_nc_3d
03669                 (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03670                 ctl->met_cloud == 2))
03671                 WARN("Cannot read cloud rain water content!");
03672             if (!read_met_nc_3d
03673                 (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03674                 ctl->met_cloud == 2))
03675                 WARN("Cannot read cloud snow water content!");
03676         }
03677     }

```

```

03677     if (ctl->met_relhum) {
03678         if (!read_met_nc_3d(ncid, "rh", "RH", ctl, met, met->h2o, 0.01f, 1))
03679             WARN("Cannot read relative humidity!");
03680 #pragma omp parallel for default(shared) collapse(2)
03681         for (int ix = 0; ix < met->nx; ix++)
03682             for (int iy = 0; iy < met->ny; iy++)
03683                 for (int ip = 0; ip < met->np; ip++) {
03684                     double pw = met->h2o[ix][iy][ip] * PSAT(met->t[ix][iy][ip]);
03685                     met->h2o[ix][iy][ip] =
03686                         (float) (pw / (met->p[ip] - (1.0 - EPS) * pw));
03687                 }
03688     }
03689
03690     /* Transfer from model levels to pressure levels... */
03691     if (ctl->met_np > 0) {
03692
03693         /* Read pressure on model levels... */
03694         if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->p1, 0.01f, 1))
03695             ERRMSG("Cannot read pressure on model levels!");
03696
03697         /* Vertical interpolation from model to pressure levels... */
03698         read_met_ml2pl(ctl, met, met->t);
03699         read_met_ml2pl(ctl, met, met->u);
03700         read_met_ml2pl(ctl, met, met->v);
03701         read_met_ml2pl(ctl, met, met->w);
03702         read_met_ml2pl(ctl, met, met->h2o);
03703         read_met_ml2pl(ctl, met, met->o3);
03704         read_met_ml2pl(ctl, met, met->lwc);
03705         read_met_ml2pl(ctl, met, met->iwc);
03706
03707         /* Set new pressure levels... */
03708         met->np = ctl->met_np;
03709         for (int ip = 0; ip < met->np; ip++)
03710             met->p[ip] = ctl->met_p[ip];
03711     }
03712 }
03713
03714 /* CLaMS meteo data... */
03715 else if (ctl->clams_met_data == 1) {
03716
03717     /* Read meteorological data... */
03718     if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
03719         ERRMSG("Cannot read temperature!");
03720     if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03721         ERRMSG("Cannot read zonal wind!");
03722     if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03723         ERRMSG("Cannot read meridional wind!");
03724     if (!read_met_nc_3d(ncid, "w", "OMEGA", ctl, met, met->w, 0.01f, 1))
03725         WARN("Cannot read vertical velocity!");
03726     if (!read_met_nc_3d(ncid, "zeta", "ZETA", ctl, met, met->zeta, 1.0, 1))
03727         WARN("Cannot read ZETA in meteo data!");
03728     if (ctl->vert_vel == 1) {
03729         if (!read_met_nc_3d
03730             (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
03731              0.00001157407f, 1)) {
03732             if (!read_met_nc_3d
03733                 (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
03734                  0.00001157407f, 1)) {
03735                 WARN("Cannot read vertical velocity!");
03736             }
03737         }
03738     }
03739
03740     if (!read_met_nc_3d
03741         (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03742         WARN("Cannot read specific humidity!");
03743     if (!read_met_nc_3d
03744         (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03745         WARN("Cannot read ozone data!");
03746     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03747         if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03748             WARN("Cannot read cloud liquid water content!");
03749         if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03750             WARN("Cannot read cloud ice water content!");
03751     }
03752     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03753         if (!read_met_nc_3d
03754             (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03755              ctl->met_cloud == 2))
03756             WARN("Cannot read cloud rain water content!");
03757         if (!read_met_nc_3d
03758             (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03759              ctl->met_cloud == 2))
03760             WARN("Cannot read cloud snow water content!");
03761     }
03762
03763     /* Transfer from model levels to pressure levels... */

```



```

03764     if (ctl->met_np > 0) {
03765
03766         /* Read pressure on model levels... */
03767         if (!read_met_nc_3d(ncid, "p1", "PRESS", ctl, met, met->p1, 1.0, 1))
03768             ERRMSG("Cannot read pressure on model levels!");
03769
03770         /* Vertical interpolation from model to pressure levels... */
03771         read_met_ml2pl(ctl, met, met->t);
03772         read_met_ml2pl(ctl, met, met->u);
03773         read_met_ml2pl(ctl, met, met->v);
03774         read_met_ml2pl(ctl, met, met->w);
03775         read_met_ml2pl(ctl, met, met->h2o);
03776         read_met_ml2pl(ctl, met, met->o3);
03777         read_met_ml2pl(ctl, met, met->lwc);
03778         read_met_ml2pl(ctl, met, met->iwc);
03779         if (ctl->vert_vel == 1) {
03780             read_met_ml2pl(ctl, met, met->zeta);
03781             read_met_ml2pl(ctl, met, met->zeta_dot);
03782         }
03783
03784         /* Set new pressure levels... */
03785         met->np = ctl->met_np;
03786         for (int ip = 0; ip < met->np; ip++)
03787             met->p[ip] = ctl->met_p[ip];
03788
03789         /* Create a pressure field... */
03790         for (int i = 0; i < met->nx; i++)
03791             for (int j = 0; j < met->ny; j++)
03792                 for (int k = 0; k < met->np; k++) {
03793                     met->patp[i][j][k] = (float) met->p[k];
03794                 }
03795     }
03796 } else
03797     ERRMSG("Meteo data format unknown!");
03798
03799 /* Check ordering of pressure levels... */
03800 for (int ip = 1; ip < met->np; ip++)
03801     if (met->p[ip - 1] < met->p[ip])
03802         ERRMSG("Pressure levels must be descending!");
03803 }
03804
03805 /*****
03806
03807 void read_met_ml2pl(
03808     ctl_t * ctl,
03809     met_t * met,
03810     float var[EX][EY][EP]) {
03811
03812     double aux[EP], p[EP];
03813
03814     /* Set timer... */
03815     SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03816     LOG(2, "Interpolate meteo data to pressure levels...");
03817
03818     /* Loop over columns... */
03819     #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03820     for (int ix = 0; ix < met->nx; ix++)
03821         for (int iy = 0; iy < met->ny; iy++) {
03822
03823             /* Copy pressure profile... */
03824             for (int ip = 0; ip < met->np; ip++)
03825                 p[ip] = met->p1[ix][iy][ip];
03826
03827             /* Interpolate... */
03828             for (int ip = 0; ip < ctl->met_np; ip++) {
03829                 double pt = ctl->met_p[ip];
03830                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03831                     pt = p[0];
03832                 else if ((pt > p[met->np - 1] && p[1] > p[0])
03833                     || (pt < p[met->np - 1] && p[1] < p[0]))
03834                     pt = p[met->np - 1];
03835                 int ip2 = locate_irr(p, met->np, pt);
03836                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03837                     p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03838             }
03839
03840             /* Copy data... */
03841             for (int ip = 0; ip < ctl->met_np; ip++)
03842                 var[ix][iy][ip] = (float) aux[ip];
03843         }
03844 }
03845
03846 /*****
03847
03848 int read_met_nc_2d(
03849     int ncid,
03850     char *varname,

```

```

03851 char *varname2,
03852 ctl_t * ctl,
03853 met_t * met,
03854 float dest[EX][EY],
03855 float scl,
03856 int init) {
03857
03858 char varsel[LEN];
03859
03860 float offset, scalfac;
03861
03862 int varid;
03863
03864 /* Check if variable exists... */
03865 if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03866     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03867         WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
03868         return 0;
03869     } else {
03870         sprintf(varsel, "%s", varname2);
03871     } else
03872         sprintf(varsel, "%s", varname);
03873
03874 /* Read packed data... */
03875 if (ctl->met_nc_scale
03876     && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03877     && nc_get_att_float(ncid, varid, "scale_factor",
03878                         &scalfac) == NC_NOERR) {
03879
03880     /* Allocate... */
03881     short *help;
03882     ALLOC(help, short,
03883           EX * EY * EP);
03884
03885     /* Read fill value and missing value... */
03886     short fillval, missval;
03887     if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03888         fillval = 0;
03889     if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03890         missval = 0;
03891
03892     /* Write info... */
03893     LOG(2, "Read 2-D variable: %s"
03894         " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03895         varsel, fillval, missval, scalfac, offset);
03896
03897     /* Read data... */
03898     NC(nc_get_var_short(ncid, varid, help));
03899
03900     /* Copy and check data... */
03901 #pragma omp parallel for default(shared) num_threads(12)
03902     for (int ix = 0; ix < met->nx; ix++)
03903         for (int iy = 0; iy < met->ny; iy++) {
03904             if (init)
03905                 dest[ix][iy] = 0;
03906             short aux = help[ARRAY_2D(iy, ix, met->nx)];
03907             if ((fillval == 0 || aux != fillval)
03908                 && (missval == 0 || aux != missval)
03909                 && fabsf(aux * scalfac + offset) < 1e14f)
03910                 dest[ix][iy] += scl * (aux * scalfac + offset);
03911             else
03912                 dest[ix][iy] = GSL_NAN;
03913         }
03914
03915     /* Free... */
03916     free(help);
03917 }
03918
03919 /* Unpacked data... */
03920 else {
03921
03922     /* Allocate... */
03923     float *help;
03924     ALLOC(help, float,
03925           EX * EY);
03926
03927     /* Read fill value and missing value... */
03928     float fillval, missval;
03929     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03930         fillval = 0;
03931     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03932         missval = 0;
03933
03934     /* Write info... */
03935     LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03936         varsel, fillval, missval);
03937

```

```

03938     /* Read data... */
03939     NC(nc_get_var_float(ncid, varid, help));
03940
03941     /* Copy and check data... */
03942 #pragma omp parallel for default(shared) num_threads(12)
03943     for (int ix = 0; ix < met->nx; ix++)
03944         for (int iy = 0; iy < met->ny; iy++) {
03945             if (init)
03946                 dest[ix][iy] = 0;
03947             float aux = help[ARRAY_2D(iy, ix, met->nx)];
03948             if ((fillval == 0 || aux != fillval)
03949                 && (missval == 0 || aux != missval)
03950                 && fabsf(aux) < 1e14f)
03951                 dest[ix][iy] += scl * aux;
03952             else
03953                 dest[ix][iy] = GSL_NAN;
03954         }
03955
03956     /* Free... */
03957     free(help);
03958 }
03959
03960 /* Return... */
03961 return 1;
03962 }
03963
03964 /*****
03965
03966 int read_met_nc_3d(
03967     int ncid,
03968     char *varname,
03969     char *varname2,
03970     ctl_t *ctl,
03971     met_t *met,
03972     float dest[EX][EY][EP],
03973     float scl,
03974     int init) {
03975
03976     char varsel[LEN];
03977
03978     float offset, scalfac;
03979
03980     int varid;
03981
03982     /* Check if variable exists... */
03983     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03984         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03985             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03986             return 0;
03987         } else {
03988             sprintf(varsel, "%s", varname2);
03989         } else
03990             sprintf(varsel, "%s", varname);
03991
03992     /* Read packed data... */
03993     if (ctl->met_nc_scale
03994         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03995         && nc_get_att_float(ncid, varid, "scale_factor",
03996                             &scalfac) == NC_NOERR) {
03997
03998         /* Allocate... */
03999         short *help;
04000         ALLOC(help, short,
04001              EX * EY * EP);
04002
04003         /* Read fill value and missing value... */
04004         short fillval, missval;
04005         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
04006             fillval = 0;
04007         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
04008             missval = 0;
04009
04010         /* Write info... */
04011         LOG(2, "Read 3-D variable: %s "
04012             " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
04013             varsel, fillval, missval, scalfac, offset);
04014
04015         /* Read data... */
04016         NC(nc_get_var_short(ncid, varid, help));
04017
04018         /* Copy and check data... */
04019 #pragma omp parallel for default(shared) num_threads(12)
04020         for (int ix = 0; ix < met->nx; ix++)
04021             for (int iy = 0; iy < met->ny; iy++)
04022                 for (int ip = 0; ip < met->np; ip++) {
04023                     if (init)
04024                         dest[ix][iy][ip] = 0;

```

```

04025     short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04026     if ((fillval == 0 || aux != fillval)
04027         && (missval == 0 || aux != missval)
04028         && fabsf(aux * scalfac + offset) < 1e14f)
04029         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
04030     else
04031         dest[ix][iy][ip] = GSL_NAN;
04032 }
04033
04034 /* Free... */
04035 free(help);
04036 }
04037
04038 /* Unpacked data... */
04039 else {
04040
04041     /* Allocate... */
04042     float *help;
04043     ALLOC(help, float,
04044           EX * EY * EP);
04045
04046     /* Read fill value and missing value... */
04047     float fillval, missval;
04048     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
04049         fillval = 0;
04050     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
04051         missval = 0;
04052
04053     /* Write info... */
04054     LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
04055         varsel, fillval, missval);
04056
04057     /* Read data... */
04058     NC(nc_get_var_float(ncid, varid, help));
04059
04060     /* Copy and check data... */
04061 #pragma omp parallel for default(shared) num_threads(12)
04062     for (int ix = 0; ix < met->nx; ix++)
04063         for (int iy = 0; iy < met->ny; iy++)
04064             for (int ip = 0; ip < met->np; ip++) {
04065                 if (init)
04066                     dest[ix][iy][ip] = 0;
04067                 float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04068                 if ((fillval == 0 || aux != fillval)
04069                     && (missval == 0 || aux != missval)
04070                     && fabsf(aux) < 1e14f)
04071                     dest[ix][iy][ip] += scl * aux;
04072                 else
04073                     dest[ix][iy][ip] = GSL_NAN;
04074             }
04075
04076     /* Free... */
04077     free(help);
04078 }
04079
04080 /* Return... */
04081 return 1;
04082 }
04083
04084 /*****
04085
04086 void read_met_pbl(
04087     met_t * met) {
04088
04089     /* Set timer... */
04090     SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
04091     LOG(2, "Calculate planetary boundary layer...");
04092
04093     /* Parameters used to estimate the height of the PBL
04094        (e.g., Vogelezang and Holtslag, 1996; Seidel et al., 2012)... */
04095     const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
04096
04097     /* Loop over grid points... */
04098 #pragma omp parallel for default(shared) collapse(2)
04099     for (int ix = 0; ix < met->nx; ix++)
04100         for (int iy = 0; iy < met->ny; iy++) {
04101
04102             /* Set bottom level of PBL... */
04103             double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
04104
04105             /* Find lowest level near the bottom... */
04106             int ip;
04107             for (ip = 1; ip < met->np; ip++)
04108                 if (met->p[ip] < pbl_bot)
04109                     break;
04110
04111             /* Get near surface data... */

```

```

04112     double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
04113                    met->p[ip], met->z[ix][iy][ip], pbl_bot);
04114     double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
04115                    met->p[ip], met->t[ix][iy][ip], pbl_bot);
04116     double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
04117                    met->p[ip], met->u[ix][iy][ip], pbl_bot);
04118     double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
04119                    met->p[ip], met->v[ix][iy][ip], pbl_bot);
04120     double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
04121                      met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
04122     double tvs = THETA_VIRT(pbl_bot, ts, h2os);
04123
04124     /* Init... */
04125     double rib_old = 0;
04126
04127     /* Loop over levels... */
04128     for (; ip < met->np; ip++) {
04129
04130         /* Get squared horizontal wind speed... */
04131         double vh2
04132             = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
04133         vh2 = GSL_MAX(vh2, SQR(umin));
04134
04135         /* Calculate bulk Richardson number... */
04136         double rib = G0 * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
04137             * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
04138                          met->h2o[ix][iy][ip]) - tvs) / vh2;
04139
04140         /* Check for critical value... */
04141         if (rib >= rib_crit) {
04142             met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
04143                                           rib, met->p[ip], rib_crit));
04144             if (met->pbl[ix][iy] > pbl_bot)
04145                 met->pbl[ix][iy] = (float) pbl_bot;
04146             break;
04147         }
04148
04149         /* Save Richardson number... */
04150         rib_old = rib;
04151     }
04152 }
04153 }
04154
04155 /*****
04156
04157 void read_met_periodic(
04158     met_t * met) {
04159
04160     /* Set timer... */
04161     SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
04162     LOG(2, "Apply periodic boundary conditions...");
04163
04164     /* Check longitudes... */
04165     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
04166              + met->lon[1] - met->lon[0] - 360) < 0.01))
04167         return;
04168
04169     /* Increase longitude counter... */
04170     if ((++met->nx) > EX)
04171         ERRMSG("Cannot create periodic boundary conditions!");
04172
04173     /* Set longitude... */
04174     met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
04175
04176     /* Loop over latitudes and pressure levels... */
04177     #pragma omp parallel for default(shared)
04178     for (int iy = 0; iy < met->ny; iy++) {
04179         met->ps[met->nx - 1][iy] = met->ps[0][iy];
04180         met->zs[met->nx - 1][iy] = met->zs[0][iy];
04181         met->ts[met->nx - 1][iy] = met->ts[0][iy];
04182         met->us[met->nx - 1][iy] = met->us[0][iy];
04183         met->vs[met->nx - 1][iy] = met->vs[0][iy];
04184         for (int ip = 0; ip < met->np; ip++) {
04185             met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
04186             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
04187             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
04188             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
04189             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
04190             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
04191             met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
04192             met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
04193         }
04194     }
04195 }
04196
04197 /*****
04198

```

```

04199 void read_met_pv(
04200     met_t * met) {
04201
04202     double pows[EP];
04203
04204     /* Set timer... */
04205     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
04206     LOG(2, "Calculate potential vorticity...");
04207
04208     /* Set powers... */
04209     #pragma omp parallel for default(shared)
04210     for (int ip = 0; ip < met->np; ip++)
04211         pows[ip] = pow(1000. / met->p[ip], 0.286);
04212
04213     /* Loop over grid points... */
04214     #pragma omp parallel for default(shared)
04215     for (int ix = 0; ix < met->nx; ix++) {
04216
04217         /* Set indices... */
04218         int ix0 = GSL_MAX(ix - 1, 0);
04219         int ix1 = GSL_MIN(ix + 1, met->nx - 1);
04220
04221         /* Loop over grid points... */
04222         for (int iy = 0; iy < met->ny; iy++) {
04223
04224             /* Set indices... */
04225             int iy0 = GSL_MAX(iy - 1, 0);
04226             int iy1 = GSL_MIN(iy + 1, met->ny - 1);
04227
04228             /* Set auxiliary variables... */
04229             double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
04230             double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
04231             double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
04232             double c0 = cos(met->lat[iy0] / 180. * M_PI);
04233             double c1 = cos(met->lat[iy1] / 180. * M_PI);
04234             double cr = cos(latr / 180. * M_PI);
04235             double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
04236
04237             /* Loop over grid points... */
04238             for (int ip = 0; ip < met->np; ip++) {
04239
04240                 /* Get gradients in longitude... */
04241                 double dtdx
04242                     = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
04243                 double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
04244
04245                 /* Get gradients in latitude... */
04246                 double dtdy
04247                     = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
04248                 double dudy
04249                     = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
04250
04251                 /* Set indices... */
04252                 int ip0 = GSL_MAX(ip - 1, 0);
04253                 int ip1 = GSL_MIN(ip + 1, met->np - 1);
04254
04255                 /* Get gradients in pressure... */
04256                 double dtdp, dudp, dvdp;
04257                 double dp0 = 100. * (met->p[ip] - met->p[ip0]);
04258                 double dp1 = 100. * (met->p[ip1] - met->p[ip]);
04259                 if (ip != ip0 && ip != ip1) {
04260                     double denom = dp0 * dp1 * (dp0 + dp1);
04261                     dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
04262                         - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
04263                         + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
04264                         / denom;
04265                     dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
04266                         - dp1 * dp1 * met->u[ix][iy][ip0]
04267                         + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
04268                         / denom;
04269                     dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
04270                         - dp1 * dp1 * met->v[ix][iy][ip0]
04271                         + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
04272                         / denom;
04273                 } else {
04274                     double denom = dp0 + dp1;
04275                     dtdp =
04276                         (met->t[ix][iy][ip1] * pows[ip1] -
04277                         met->t[ix][iy][ip0] * pows[ip0]) / denom;
04278                     dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
04279                     dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
04280                 }
04281
04282                 /* Calculate PV... */
04283                 met->pv[ix][iy][ip] = (float)
04284                     (1e6 * G0 *
04285                     (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));

```

```

04286     }
04287     }
04288 }
04289
04290 /* Fix for polar regions... */
04291 #pragma omp parallel for default(shared)
04292 for (int ix = 0; ix < met->nx; ix++)
04293     for (int ip = 0; ip < met->np; ip++) {
04294         met->pv[ix][0][ip]
04295         = met->pv[ix][1][ip]
04296         = met->pv[ix][2][ip];
04297         met->pv[ix][met->ny - 1][ip]
04298         = met->pv[ix][met->ny - 2][ip]
04299         = met->pv[ix][met->ny - 3][ip];
04300     }
04301 }
04302
04303 /*****
04304 void read_met_sample(
04305     ctl_t * ctl,
04306     met_t * met) {
04307
04308     met_t *help;
04309
04310     /* Check parameters... */
04311     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
04312         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
04313         return;
04314
04315     /* Set timer... */
04316     SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
04317     LOG(2, "Downsampling of meteo data...");
04318
04319     /* Allocate... */
04320     ALLOC(help, met_t, 1);
04321
04322     /* Copy data... */
04323     help->nx = met->nx;
04324     help->ny = met->ny;
04325     help->np = met->np;
04326     memcpy(help->lon, met->lon, sizeof(met->lon));
04327     memcpy(help->lat, met->lat, sizeof(met->lat));
04328     memcpy(help->p, met->p, sizeof(met->p));
04329
04330     /* Smoothing... */
04331     for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
04332         for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
04333             for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
04334                 help->ps[ix][iy] = 0;
04335                 help->zs[ix][iy] = 0;
04336                 help->ts[ix][iy] = 0;
04337                 help->us[ix][iy] = 0;
04338                 help->vs[ix][iy] = 0;
04339                 help->t[ix][iy][ip] = 0;
04340                 help->u[ix][iy][ip] = 0;
04341                 help->v[ix][iy][ip] = 0;
04342                 help->w[ix][iy][ip] = 0;
04343                 help->h2o[ix][iy][ip] = 0;
04344                 help->o3[ix][iy][ip] = 0;
04345                 help->lwc[ix][iy][ip] = 0;
04346                 help->iwc[ix][iy][ip] = 0;
04347                 float wsum = 0;
04348                 for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
04349                     ix2++) {
04350                     int ix3 = ix2;
04351                     if (ix3 < 0)
04352                         ix3 += met->nx;
04353                     else if (ix3 >= met->nx)
04354                         ix3 -= met->nx;
04355
04356                     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
04357                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
04358                         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
04359                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
04360                             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
04361                                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
04362                                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
04363                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
04364                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
04365                             help->ts[ix][iy] += w * met->ts[ix3][iy2];
04366                             help->us[ix][iy] += w * met->us[ix3][iy2];
04367                             help->vs[ix][iy] += w * met->vs[ix3][iy2];
04368                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
04369                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
04370                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
04371                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
04372

```

```

04373         help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
04374         help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
04375         help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
04376         help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
04377         wsum += w;
04378     }
04379 }
04380 help->ps[ix][iy] /= wsum;
04381 help->zs[ix][iy] /= wsum;
04382 help->ts[ix][iy] /= wsum;
04383 help->us[ix][iy] /= wsum;
04384 help->vs[ix][iy] /= wsum;
04385 help->t[ix][iy][ip] /= wsum;
04386 help->u[ix][iy][ip] /= wsum;
04387 help->v[ix][iy][ip] /= wsum;
04388 help->w[ix][iy][ip] /= wsum;
04389 help->h2o[ix][iy][ip] /= wsum;
04390 help->o3[ix][iy][ip] /= wsum;
04391 help->lwc[ix][iy][ip] /= wsum;
04392 help->iwc[ix][iy][ip] /= wsum;
04393 }
04394 }
04395 }
04396
04397 /* Downsampling... */
04398 met->nx = 0;
04399 for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04400     met->lon[met->nx] = help->lon[ix];
04401     met->ny = 0;
04402     for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04403         met->lat[met->ny] = help->lat[iy];
04404         met->ps[met->nx][met->ny] = help->ps[ix][iy];
04405         met->zs[met->nx][met->ny] = help->zs[ix][iy];
04406         met->ts[met->nx][met->ny] = help->ts[ix][iy];
04407         met->us[met->nx][met->ny] = help->us[ix][iy];
04408         met->vs[met->nx][met->ny] = help->vs[ix][iy];
04409         met->np = 0;
04410         for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04411             met->p[met->np] = help->p[ip];
04412             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04413             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04414             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04415             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04416             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04417             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04418             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
04419             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04420             met->np++;
04421         }
04422         met->ny++;
04423     }
04424     met->nx++;
04425 }
04426
04427 /* Free... */
04428 free(help);
04429 }
04430
04431 /*****
04432
04433 void read_met_surface(
04434     int ncid,
04435     met_t * met,
04436     ctl_t * ctl) {
04437
04438     /* Set timer... */
04439     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04440     LOG(2, "Read surface data...");
04441
04442     /* MPTRAC meteo data... */
04443     if (ctl->clams_met_data == 0) {
04444
04445         /* Read surface pressure... */
04446         if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
04447             if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04448                 if (!read_met_nc_2d(ncid, "sp", "SP", ctl, met, met->ps, 0.01f, 1)) {
04449                     WARN("Cannot not read surface pressure data (use lowest level)!");
04450                     for (int ix = 0; ix < met->nx; ix++)
04451                         for (int iy = 0; iy < met->ny; iy++)
04452                             met->ps[ix][iy] = (float) met->p[0];
04453                 }
04454             }
04455         } else
04456             for (int ix = 0; ix < met->nx; ix++)
04457                 for (int iy = 0; iy < met->ny; iy++)
04458                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04459

```



```

04460      /* Read geopotential height at the surface... */
04461      if (!read_met_nc_2d
04462          (ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
04463          if (!read_met_nc_2d
04464              (ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
04465              WARN("Cannot read surface geopotential height!");
04466
04467      /* Read temperature at the surface... */
04468      if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
04469          WARN("Cannot read surface temperature!");
04470
04471      /* Read zonal wind at the surface... */
04472      if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
04473          WARN("Cannot read surface zonal wind!");
04474
04475      /* Read meridional wind at the surface... */
04476      if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
04477          WARN("Cannot read surface meridional wind!");
04478  }
04479
04480  /* CLaMS meteo data... */
04481  else {
04482
04483      /* Read surface pressure... */
04484      if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04485          WARN("Cannot not read surface pressure data (use lowest level!)");
04486          for (int ix = 0; ix < met->nx; ix++)
04487              for (int iy = 0; iy < met->ny; iy++)
04488                  met->ps[ix][iy] = (float) met->p[0];
04489      }
04490
04491      /* Read geopotential height at the surface
04492         (use lowermost level of 3-D data field)... */
04493      float *help;
04494      ALLOC(help, float,
04495            EX * EY * EP);
04496      memcpy(help, met->p1, sizeof(met->p1));
04497      if (!read_met_nc_3d
04498          (ncid, "gph", "GPH", ctl, met, met->p1, (float) (1e-3 / G0), 1)) {
04499          ERRMSG("Cannot read geopotential height!");
04500      } else
04501          for (int ix = 0; ix < met->nx; ix++)
04502              for (int iy = 0; iy < met->ny; iy++)
04503                  met->zs[ix][iy] = met->p1[ix][iy][0];
04504      memcpy(met->p1, help, sizeof(met->p1));
04505      free(help);
04506
04507      /* Read temperature at the surface... */
04508      if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
04509          WARN("Cannot read surface temperature!");
04510
04511      /* Read zonal wind at the surface... */
04512      if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
04513          WARN("Cannot read surface zonal wind!");
04514
04515      /* Read meridional wind at the surface... */
04516      if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
04517          WARN("Cannot read surface meridional wind!");
04518  }
04519 }
04520
04521 /*****
04522
04523 void read_met_tropo(
04524     ctl_t * ctl,
04525     clim_t * clim,
04526     met_t * met) {
04527
04528     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04529           th2[200], z[EP], z2[200];
04530
04531     /* Set timer... */
04532     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04533     LOG(2, "Calculate tropopause...");
04534
04535     /* Get altitude and pressure profiles... */
04536     #pragma omp parallel for default(shared)
04537     for (int iz = 0; iz < met->np; iz++)
04538         z[iz] = Z(met->p[iz]);
04539     #pragma omp parallel for default(shared)
04540     for (int iz = 0; iz <= 190; iz++) {
04541         z2[iz] = 4.5 + 0.1 * iz;
04542         p2[iz] = P(z2[iz]);
04543     }
04544
04545     /* Do not calculate tropopause... */
04546     if (ctl->met_tropo == 0)

```

```

04547 #pragma omp parallel for default(shared) collapse(2)
04548     for (int ix = 0; ix < met->nx; ix++)
04549         for (int iy = 0; iy < met->ny; iy++)
04550             met->pt[ix][iy] = GSL_NAN;
04551
04552     /* Use tropopause climatology... */
04553     else if (ctl->met_tropo == 1) {
04554 #pragma omp parallel for default(shared) collapse(2)
04555         for (int ix = 0; ix < met->nx; ix++)
04556             for (int iy = 0; iy < met->ny; iy++)
04557                 met->pt[ix][iy] = (float) clim_tropo(clim, met->time, met->lat[iy]);
04558     }
04559
04560     /* Use cold point... */
04561     else if (ctl->met_tropo == 2) {
04562
04563         /* Loop over grid points... */
04564 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04565         for (int ix = 0; ix < met->nx; ix++)
04566             for (int iy = 0; iy < met->ny; iy++) {
04567
04568                 /* Interpolate temperature profile... */
04569                 for (int iz = 0; iz < met->np; iz++)
04570                     t[iz] = met->t[ix][iy][iz];
04571                 spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04572
04573                 /* Find minimum... */
04574                 int iz = (int) gsl_stats_min_index(t2, 1, 171);
04575                 if (iz > 0 && iz < 170)
04576                     met->pt[ix][iy] = (float) p2[iz];
04577                 else
04578                     met->pt[ix][iy] = GSL_NAN;
04579             }
04580     }
04581
04582     /* Use WMO definition... */
04583     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04584
04585         /* Loop over grid points... */
04586 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04587         for (int ix = 0; ix < met->nx; ix++)
04588             for (int iy = 0; iy < met->ny; iy++) {
04589
04590                 /* Interpolate temperature profile... */
04591                 int iz;
04592                 for (iz = 0; iz < met->np; iz++)
04593                     t[iz] = met->t[ix][iy][iz];
04594                 spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04595
04596                 /* Find 1st tropopause... */
04597                 met->pt[ix][iy] = GSL_NAN;
04598                 for (iz = 0; iz <= 170; iz++) {
04599                     int found = 1;
04600                     for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04601                         if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04602                             ctl->met_tropo_lapse) {
04603                             found = 0;
04604                             break;
04605                         }
04606                     if (found) {
04607                         if (iz > 0 && iz < 170)
04608                             met->pt[ix][iy] = (float) p2[iz];
04609                         break;
04610                     }
04611                 }
04612
04613                 /* Find 2nd tropopause... */
04614                 if (ctl->met_tropo == 4) {
04615                     met->pt[ix][iy] = GSL_NAN;
04616                     for (; iz <= 170; iz++) {
04617                         int found = 1;
04618                         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04619                             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04620                                 ctl->met_tropo_lapse_sep) {
04621                                 found = 0;
04622                                 break;
04623                             }
04624                         if (found)
04625                             break;
04626                     }
04627                     for (; iz <= 170; iz++) {
04628                         int found = 1;
04629                         for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04630                             if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04631                                 ctl->met_tropo_lapse) {
04632                                 found = 0;
04633                                 break;

```

```

04634         }
04635         if (found) {
04636             if (iz > 0 && iz < 170)
04637                 met->pt[ix][iy] = (float) p2[iz];
04638             break;
04639         }
04640     }
04641 }
04642 }
04643 }
04644
04645 /* Use dynamical tropopause... */
04646 else if (ctl->met_tropo == 5) {
04647
04648     /* Loop over grid points... */
04649 #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04650     for (int ix = 0; ix < met->nx; ix++)
04651         for (int iy = 0; iy < met->ny; iy++) {
04652
04653             /* Interpolate potential vorticity profile... */
04654             for (int iz = 0; iz < met->np; iz++)
04655                 pv[iz] = met->pv[ix][iy][iz];
04656             spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04657
04658             /* Interpolate potential temperature profile... */
04659             for (int iz = 0; iz < met->np; iz++)
04660                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04661             spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04662
04663             /* Find dynamical tropopause... */
04664             met->pt[ix][iy] = GSL_NAN;
04665             for (int iz = 0; iz <= 170; iz++)
04666                 if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04667                     || th2[iz] >= ctl->met_tropo_theta) {
04668                     if (iz > 0 && iz < 170)
04669                         met->pt[ix][iy] = (float) p2[iz];
04670                     break;
04671                 }
04672         }
04673     }
04674
04675     else
04676         ERRMSG("Cannot calculate tropopause!");
04677
04678     /* Interpolate temperature, geopotential height, and water vapor vmr... */
04679 #pragma omp parallel for default(shared) collapse(2)
04680     for (int ix = 0; ix < met->nx; ix++)
04681         for (int iy = 0; iy < met->ny; iy++) {
04682             double h2ot, tt, zt;
04683             INTPOL_INIT;
04684             intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04685                               met->lat[iy], &tt, ci, cw, 1);
04686             intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04687                               met->lat[iy], &zt, ci, cw, 0);
04688             intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04689                               met->lat[iy], &h2ot, ci, cw, 0);
04690             met->tt[ix][iy] = (float) tt;
04691             met->zt[ix][iy] = (float) zt;
04692             met->h2ot[ix][iy] = (float) h2ot;
04693         }
04694 }
04695
04696 /*****
04697
04698 void read_obs(
04699     char *filename,
04700     double *rt,
04701     double *rz,
04702     double *rlon,
04703     double *rlat,
04704     double *robs,
04705     int *nobs) {
04706
04707     FILE *in;
04708
04709     char line[LEN];
04710
04711     /* Open observation data file... */
04712     LOG(1, "Read observation data: %s", filename);
04713     if (!(in = fopen(filename, "r")))
04714         ERRMSG("Cannot open file!");
04715
04716     /* Read observations... */
04717     while (fgets(line, LEN, in))
04718         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
04719                 &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
04720             if ((++(*nobs)) >= NOBS)

```

```

04721         ERRMSG("Too many observations!");
04722
04723     /* Close observation data file... */
04724     fclose(in);
04725
04726     /* Check time... */
04727     for (int i = 1; i < *nobs; i++)
04728         if (rt[i] < rt[i - 1])
04729             ERRMSG("Time must be ascending!");
04730
04731     /* Write info... */
04732     int n = *nobs;
04733     double mini, maxi;
04734     LOG(2, "Number of observations: %d", *nobs);
04735     gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
04736     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04737     gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
04738     LOG(2, "Altitude range: %g ... %g km", mini, maxi);
04739     gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
04740     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04741     gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
04742     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04743     gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
04744     LOG(2, "Observation range: %g ... %g", mini, maxi);
04745 }
04746
04747 /*****
04748
04749 double scan_ctl(
04750     const char *filename,
04751     int argc,
04752     char *argv[],
04753     const char *varname,
04754     int arridx,
04755     const char *defvalue,
04756     char *value) {
04757
04758     FILE *in = NULL;
04759
04760     char fullname1[LEN], fullname2[LEN], rval[LEN];
04761
04762     int contain = 0, i;
04763
04764     /* Open file... */
04765     if (filename[strlen(filename) - 1] != '-')
04766         if (!(in = fopen(filename, "r")))
04767             ERRMSG("Cannot open file!");
04768
04769     /* Set full variable name... */
04770     if (arridx >= 0) {
04771         sprintf(fullname1, "%s[%d]", varname, arridx);
04772         sprintf(fullname2, "%s[*]", varname);
04773     } else {
04774         sprintf(fullname1, "%s", varname);
04775         sprintf(fullname2, "%s", varname);
04776     }
04777
04778     /* Read data... */
04779     if (in != NULL) {
04780         char dummy[LEN], line[LEN], rvarname[LEN];
04781         while (fgets(line, LEN, in)) {
04782             if (sscanf(line, "%4999s %4999s %4999s", rvarname, dummy, rval) == 3)
04783                 if (strcasecmp(rvarname, fullname1) == 0 ||
04784                     strcasecmp(rvarname, fullname2) == 0) {
04785                     contain = 1;
04786                     break;
04787                 }
04788         }
04789     }
04790     for (i = 1; i < argc - 1; i++)
04791         if (strcasecmp(argv[i], fullname1) == 0 ||
04792             strcasecmp(argv[i], fullname2) == 0) {
04793             sprintf(rval, "%s", argv[i + 1]);
04794             contain = 1;
04795             break;
04796         }
04797
04798     /* Close file... */
04799     if (in != NULL)
04800         fclose(in);
04801
04802     /* Check for missing variables... */
04803     if (!contain) {
04804         if (strlen(defvalue) > 0)
04805             sprintf(rval, "%s", defvalue);
04806         else
04807             ERRMSG("Missing variable %s!\n", fullname1);

```

```

04808     }
04809
04810     /* Write info... */
04811     LOG(1, "%s = %s", fullnamel, rval);
04812
04813     /* Return values... */
04814     if (value != NULL)
04815         sprintf(value, "%s", rval);
04816     return atof(rval);
04817 }
04818
04819 /*****
04820
04821 double sedi(
04822     double p,
04823     double T,
04824     double rp,
04825     double rhop) {
04826
04827     /* Convert particle radius from microns to m... */
04828     rp *= 1e-6;
04829
04830     /* Density of dry air [kg / m^3]... */
04831     double rho = RHO(p, T);
04832
04833     /* Dynamic viscosity of air [kg / (m s)]... */
04834     double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04835
04836     /* Thermal velocity of an air molecule [m / s]... */
04837     double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04838
04839     /* Mean free path of an air molecule [m]... */
04840     double lambda = 2. * eta / (rho * v);
04841
04842     /* Knudsen number for air (dimensionless)... */
04843     double K = lambda / rp;
04844
04845     /* Cunningham slip-flow correction (dimensionless)... */
04846     double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04847
04848     /* Sedimentation velocity [m / s]... */
04849     return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
04850 }
04851
04852 /*****
04853
04854 void spline(
04855     double *x,
04856     double *y,
04857     int n,
04858     double *x2,
04859     double *y2,
04860     int n2,
04861     int method) {
04862
04863     /* Cubic spline interpolation... */
04864     if (method == 1) {
04865
04866         /* Allocate... */
04867         gsl_interp_accel *acc;
04868         gsl_spline *s;
04869         acc = gsl_interp_accel_alloc();
04870         s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04871
04872         /* Interpolate profile... */
04873         gsl_spline_init(s, x, y, (size_t) n);
04874         for (int i = 0; i < n2; i++)
04875             if (x2[i] <= x[0])
04876                 y2[i] = y[0];
04877             else if (x2[i] >= x[n - 1])
04878                 y2[i] = y[n - 1];
04879             else
04880                 y2[i] = gsl_spline_eval(s, x2[i], acc);
04881
04882         /* Free... */
04883         gsl_spline_free(s);
04884         gsl_interp_accel_free(acc);
04885     }
04886
04887     /* Linear interpolation... */
04888     else {
04889         for (int i = 0; i < n2; i++)
04890             if (x2[i] <= x[0])
04891                 y2[i] = y[0];
04892             else if (x2[i] >= x[n - 1])
04893                 y2[i] = y[n - 1];
04894             else {

```

```

04895         int idx = locate_irr(x, n, x2[i]);
04896         y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04897     }
04898 }
04899 }
04900
04901 /*****
04902
04903 float stddev(
04904     float *data,
04905     int n) {
04906
04907     if (n <= 0)
04908         return 0;
04909
04910     float mean = 0, var = 0;
04911
04912     for (int i = 0; i < n; ++i) {
04913         mean += data[i];
04914         var += SQR(data[i]);
04915     }
04916
04917     var = var / (float) n - SQR(mean / (float) n);
04918
04919     return (var > 0 ? sqrtf(var) : 0);
04920 }
04921
04922 /*****
04923
04924 double sza(
04925     double sec,
04926     double lon,
04927     double lat) {
04928
04929     double D, dec, e, g, GMST, h, L, LST, q, ra;
04930
04931     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
04932     D = sec / 86400 - 0.5;
04933
04934     /* Geocentric apparent ecliptic longitude [rad]... */
04935     g = (357.529 + 0.98560028 * D) * M_PI / 180;
04936     q = 280.459 + 0.98564736 * D;
04937     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
04938
04939     /* Mean obliquity of the ecliptic [rad]... */
04940     e = (23.439 - 0.00000036 * D) * M_PI / 180;
04941
04942     /* Declination [rad]... */
04943     dec = asin(sin(e) * sin(L));
04944
04945     /* Right ascension [rad]... */
04946     ra = atan2(cos(e) * sin(L), cos(L));
04947
04948     /* Greenwich Mean Sidereal Time [h]... */
04949     GMST = 18.697374558 + 24.06570982441908 * D;
04950
04951     /* Local Sidereal Time [h]... */
04952     LST = GMST + lon / 15;
04953
04954     /* Hour angle [rad]... */
04955     h = LST / 12 * M_PI - ra;
04956
04957     /* Convert latitude... */
04958     lat *= M_PI / 180;
04959
04960     /* Return solar zenith angle [rad]... */
04961     return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
04962 }
04963
04964 /*****
04965
04966 void time2jsec(
04967     int year,
04968     int mon,
04969     int day,
04970     int hour,
04971     int min,
04972     int sec,
04973     double remain,
04974     double *jsec) {
04975
04976     struct tm t0, t1;
04977
04978     t0.tm_year = 100;
04979     t0.tm_mon = 0;
04980     t0.tm_mday = 1;
04981     t0.tm_hour = 0;

```

```

04982     t0.tm_min = 0;
04983     t0.tm_sec = 0;
04984
04985     t1.tm_year = year - 1900;
04986     t1.tm_mon = mon - 1;
04987     t1.tm_mday = day;
04988     t1.tm_hour = hour;
04989     t1.tm_min = min;
04990     t1.tm_sec = sec;
04991
04992     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04993 }
04994
04995 /*****
04996
04997 void timer(
04998     const char *name,
04999     const char *group,
05000     int output) {
05001
05002     static char names[NTIMER][100], groups[NTIMER][100];
05003
05004     static double rt_name[NTIMER], rt_group[NTIMER],
05005         rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;
05006
05007     static int iname = -1, igrup = -1, nname, ngroup, ct_name[NTIMER];
05008
05009     /* Get time... */
05010     t1 = omp_get_wtime();
05011     dt = t1 - t0;
05012
05013     /* Add elapsed time to current timers... */
05014     if (iname >= 0) {
05015         rt_name[iname] += dt;
05016         rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
05017         rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
05018         ct_name[iname]++;
05019     }
05020     if (igrup >= 0)
05021         rt_group[igrup] += t1 - t0;
05022
05023     /* Report timers... */
05024     if (output) {
05025         for (int i = 0; i < nname; i++)
05026             LOG(1, "TIMER_%s = %.3f s (min= %g s, mean= %g s, "
05027                 " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
05028                 rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
05029         for (int i = 0; i < ngroup; i++)
05030             LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
05031         double total = 0.0;
05032         for (int i = 0; i < nname; i++)
05033             total += rt_name[i];
05034         LOG(1, "TIMER_TOTAL = %.3f s", total);
05035     }
05036
05037     /* Identify IDs of next timer... */
05038     for (iname = 0; iname < nname; iname++)
05039         if (strcasecmp(name, names[iname]) == 0)
05040             break;
05041     for (igrup = 0; igrup < ngroup; igrup++)
05042         if (strcasecmp(group, groups[igrup]) == 0)
05043             break;
05044
05045     /* Check whether this is a new timer... */
05046     if (iname >= nname) {
05047         sprintf(names[iname], "%s", name);
05048         if ((++nname) > NTIMER)
05049             ERRMSG("Too many timers!");
05050     }
05051
05052     /* Check whether this is a new group... */
05053     if (igrup >= ngroup) {
05054         sprintf(groups[igrup], "%s", group);
05055         if ((++ngroup) > NTIMER)
05056             ERRMSG("Too many groups!");
05057     }
05058
05059     /* Save starting time... */
05060     t0 = t1;
05061 }
05062
05063 /*****
05064
05065 double tropo_weight(
05066     clim_t * clim,
05067     double t,
05068     double lat,

```

```

05069 double p) {
05070
05071 /* Get tropopause pressure... */
05072 double pt = clim_tropo(clim, t, lat);
05073
05074 /* Get pressure range... */
05075 double p1 = pt * 0.866877899;
05076 double p0 = pt / 0.866877899;
05077
05078 /* Get weighting factor... */
05079 if (p > p0)
05080     return 1;
05081 else if (p < p1)
05082     return 0;
05083 else
05084     return LIN(p0, 1.0, p1, 0.0, p);
05085 }
05086
05087 /*****
05088
05089 void write_atm(
05090     const char *filename,
05091     ctl_t * ctl,
05092     atm_t * atm,
05093     double t) {
05094
05095 /* Set timer... */
05096 SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
05097
05098 /* Write info... */
05099 LOG(1, "Write atmospheric data: %s", filename);
05100
05101 /* Write ASCII data... */
05102 if (ctl->atm_type == 0)
05103     write_atm_asc(filename, ctl, atm, t);
05104
05105 /* Write binary data... */
05106 else if (ctl->atm_type == 1)
05107     write_atm_bin(filename, ctl, atm);
05108
05109 /* Write netCDF data... */
05110 else if (ctl->atm_type == 2)
05111     write_atm_nc(filename, ctl, atm);
05112
05113 /* Write CLaMS data... */
05114 else if (ctl->atm_type == 3)
05115     write_atm_clams(ctl, atm, t);
05116
05117 /* Error... */
05118 else
05119     ERRMSG("Atmospheric data type not supported!");
05120
05121 /* Write info... */
05122 double mini, maxi;
05123 LOG(2, "Number of particles: %d", atm->np);
05124 gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
05125 LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
05126 gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
05127 LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
05128 LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
05129 gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
05130 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
05131 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
05132 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
05133 for (int iq = 0; iq < ctl->nq; iq++) {
05134     char msg[LEN];
05135     sprintf(msg, "Quantity %s range: %s ... %s %s",
05136             ctl->qnt_name[iq], ctl->qnt_format[iq],
05137             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
05138     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
05139     LOG(2, msg, mini, maxi);
05140 }
05141 }
05142
05143 /*****
05144
05145 void write_atm_asc(
05146     const char *filename,
05147     ctl_t * ctl,
05148     atm_t * atm,
05149     double t) {
05150
05151 FILE *out;
05152
05153 /* Set time interval for output... */
05154 double t0 = t - 0.5 * ctl->dt_mod;
05155 double t1 = t + 0.5 * ctl->dt_mod;

```



```

05156
05157 /* Check if gnuplot output is requested... */
05158 if (ctl->atm_gpfile[0] != '-') {
05159
05160     /* Create gnuplot pipe... */
05161     if (!(out = popen("gnuplot", "w")))
05162         ERRMSG("Cannot create pipe to gnuplot!");
05163
05164     /* Set plot filename... */
05165     fprintf(out, "set out \"%s.png\\n\", filename);
05166
05167     /* Set time string... */
05168     double r;
05169     int year, mon, day, hour, min, sec;
05170     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05171     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\\n\",
05172             year, mon, day, hour, min);
05173
05174     /* Dump gnuplot file to pipe... */
05175     FILE *in;
05176     if (!(in = fopen(ctl->atm_gpfile, "r")))
05177         ERRMSG("Cannot open file!");
05178     char line[LEN];
05179     while (fgets(line, LEN, in))
05180         fprintf(out, "%s", line);
05181     fclose(in);
05182 }
05183
05184 else {
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" "# $4 = latitude [deg]\\n");
05196 for (int iq = 0; iq < ctl->nq; iq++)
05197     fprintf(out, "# $%i = %s [%s]\\n", iq + 5, ctl->qnt_name[iq],
05198             ctl->qnt_unit[iq]);
05199 fprintf(out, "\\n");
05200
05201 /* Write data... */
05202 for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
05203
05204     /* Check time... */
05205     if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05206         continue;
05207
05208     /* Write output... */
05209     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
05210             atm->lon[ip], atm->lat[ip]);
05211     for (int iq = 0; iq < ctl->nq; iq++) {
05212         fprintf(out, " ");
05213         if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05214             fprintf(out, ctl->qnt_format[iq], GSL_NAN);
05215         else
05216             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05217     }
05218     fprintf(out, "\\n");
05219 }
05220
05221 /* Close file... */
05222 fclose(out);
05223 }
05224
05225 /*****
05226
05227 void write_atm_bin(
05228     const char *filename,
05229     ctl_t *ctl,
05230     atm_t *atm) {
05231
05232     FILE *out;
05233
05234     /* Create file... */
05235     if (!(out = fopen(filename, "w")))
05236         ERRMSG("Cannot create file!");
05237
05238     /* Write version of binary data... */
05239     int version = 100;
05240     FWRITE(&version, int,
05241           1,
05242           out);

```

```

05243
05244 /* Write data... */
05245 FWRITE(&atm->np, int,
05246     1,
05247     out);
05248 FWRITE(atm->time, double,
05249     (size_t) atm->np,
05250     out);
05251 FWRITE(atm->p, double,
05252     (size_t) atm->np,
05253     out);
05254 FWRITE(atm->lon, double,
05255     (size_t) atm->np,
05256     out);
05257 FWRITE(atm->lat, double,
05258     (size_t) atm->np,
05259     out);
05260 for (int iq = 0; iq < ctl->nq; iq++)
05261     FWRITE(atm->q[iq], double,
05262         (size_t) atm->np,
05263         out);
05264
05265 /* Write final flag... */
05266 int final = 999;
05267 FWRITE(&final, int,
05268     1,
05269     out);
05270
05271 /* Close file... */
05272 fclose(out);
05273 }
05274
05275 /*****
05276
05277 void write_atm_clams(
05278     ctl_t * ctl,
05279     atm_t * atm,
05280     double t) {
05281
05282     /* Global Counter... */
05283     static size_t out_cnt = 0;
05284
05285     char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";
05286
05287     double r, r_start, r_stop;
05288
05289     int year, mon, day, hour, min, sec;
05290     int year_start, mon_start, day_start, hour_start, min_start, sec_start;
05291     int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
05292     int ncid, varid, tid, pid, cid, zid, dim_ids[2];
05293
05294     /* time, nparc */
05295     size_t start[2], count[2];
05296
05297     /* Determine start and stop times of calculation... */
05298     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05299     jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
05300         &min_start, &sec_start, &r_start);
05301     jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
05302         &min_stop, &sec_stop, &r_stop);
05303
05304     /* Set filename... */
05305     sprintf(filename_out,
05306         "./traj_fix_3d_%02d%02d%02d%02d_%02d%02d%02d%02d.nc",
05307         year_start % 100, mon_start, day_start, hour_start,
05308         year_stop % 100, mon_stop, day_stop, hour_stop);
05309     printf("Write traj file: %s\n", filename_out);
05310
05311     /* Define hyperslap for the traj_file... */
05312     start[0] = out_cnt;
05313     start[1] = 0;
05314     count[0] = 1;
05315     count[1] = (size_t) atm->np;
05316
05317     /* Create the file at the first timestep... */
05318     if (out_cnt == 0) {
05319
05320         /* Create file... */
05321         nc_create(filename_out, NC_CLOBBER, &ncid);
05322
05323         /* Define dimensions... */
05324         NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
05325         NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05326         NC(nc_def_dim(ncid, "TMDT", 7, &cid));
05327         dim_ids[0] = tid;
05328         dim_ids[1] = pid;
05329

```

```

05330      /* Define variables and their attributes... */
05331      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05332                "seconds since 2000-01-01 00:00:00 UTC");
05333      NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
05334      NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
05335      NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
05336      NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
05337      for (int iq = 0; iq < ctl->nq; iq++)
05338          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05339                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05340
05341      /* Define global attributes... */
05342      NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
05343      NC_PUT_ATT_GLOBAL("model", "MPTRAC");
05344
05345      /* End definitions... */
05346      NC(nc_enddef(ncid));
05347      NC(nc_close(ncid));
05348  }
05349
05350      /* Increment global counter to change hyperslap... */
05351      out_cnt++;
05352
05353      /* Open file... */
05354      NC(nc_open(filename_out, NC_WRITE, &ncid));
05355
05356      /* Write data... */
05357      NC_PUT_DOUBLE("time", atm->time, 1);
05358      NC_PUT_DOUBLE("LAT", atm->lat, 1);
05359      NC_PUT_DOUBLE("LON", atm->lon, 1);
05360      NC_PUT_DOUBLE("PRESS", atm->p, 1);
05361      if (ctl->vert_coord_ap == 1) {
05362          NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
05363      } else if (ctl->qnt_zeta >= 0) {
05364          NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
05365      }
05366      for (int iq = 0; iq < ctl->nq; iq++)
05367          NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);
05368
05369      /* Close file... */
05370      NC(nc_close(ncid));
05371
05372      /* At the last time step create the init_fix_YYYYMMDDHH file... */
05373      if ((year == year_stop) && (mon == mon_stop)
05374          && (day == day_stop) && (hour == hour_stop)) {
05375
05376          /* Set filename... */
05377          char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
05378          sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
05379                  year_stop % 100, mon_stop, day_stop, hour_stop);
05380          printf("Write init file: %s\n", filename_init);
05381
05382          /* Create file... */
05383          nc_create(filename_init, NC_CLOBBER, &ncid);
05384
05385          /* Define dimensions... */
05386          NC(nc_def_dim(ncid, "time", 1, &tid));
05387          NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05388          dim_ids[0] = tid;
05389          dim_ids[1] = pid;
05390
05391          /* Define variables and their attributes... */
05392          NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05393                    "seconds since 2000-01-01 00:00:00 UTC");
05394          NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
05395          NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
05396          NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
05397          NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
05398          NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
05399          NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
05400          for (int iq = 0; iq < ctl->nq; iq++)
05401              NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05402                        ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05403
05404          /* Define global attributes... */
05405          NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
05406          NC_PUT_ATT_GLOBAL("model", "MPTRAC");
05407
05408          /* End definitions... */
05409          NC(nc_enddef(ncid));
05410
05411          /* Write data... */
05412          NC_PUT_DOUBLE("time", atm->time, 0);
05413          NC_PUT_DOUBLE("LAT", atm->lat, 0);
05414          NC_PUT_DOUBLE("LON", atm->lon, 0);
05415          NC_PUT_DOUBLE("PRESS", atm->p, 0);
05416          NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);

```

```

05417     for (int iq = 0; iq < ctl->nq; iq++)
05418         NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05419
05420     /* Close file... */
05421     NC(nc_close(ncid));
05422 }
05423 }
05424
05425 /*****
05426
05427 void write_atm_nc(
05428     const char *filename,
05429     ctl_t * ctl,
05430     atm_t * atm) {
05431
05432     int ncid, obsid, varid;
05433
05434     size_t start[2], count[2];
05435
05436     /* Create file... */
05437     nc_create(filename, NC_CLOBBER, &ncid);
05438
05439     /* Define dimensions... */
05440     NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));
05441
05442     /* Define variables and their attributes... */
05443     NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
05444         "seconds since 2000-01-01 00:00:00 UTC");
05445     NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");
05446     NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
05447     NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
05448     for (int iq = 0; iq < ctl->nq; iq++)
05449         NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
05450             ctl->qnt_longname[iq], ctl->qnt_unit[iq]);
05451
05452     /* Define global attributes... */
05453     NC_PUT_ATT_GLOBAL("featureType", "point");
05454
05455     /* End definitions... */
05456     NC(nc_enddef(ncid));
05457
05458     /* Write data... */
05459     NC_PUT_DOUBLE("time", atm->time, 0);
05460     NC_PUT_DOUBLE("press", atm->p, 0);
05461     NC_PUT_DOUBLE("lon", atm->lon, 0);
05462     NC_PUT_DOUBLE("lat", atm->lat, 0);
05463     for (int iq = 0; iq < ctl->nq; iq++)
05464         NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05465
05466     /* Close file... */
05467     NC(nc_close(ncid));
05468 }
05469
05470 /*****
05471
05472 void write_csi(
05473     const char *filename,
05474     ctl_t * ctl,
05475     atm_t * atm,
05476     double t) {
05477
05478     static FILE *out;
05479
05480     static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
05481         dlon, dlat, dz, x[NCSI], y[NCSI];
05482
05483     static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nobs;
05484
05485     /* Set timer... */
05486     SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
05487
05488     /* Init... */
05489     if (t == ctl->t_start) {
05490
05491         /* Check quantity index for mass... */
05492         if (ctl->qnt_m < 0)
05493             ERRMSG("Need quantity mass!");
05494
05495         /* Allocate... */
05496         ALLOC(area, double,
05497             ctl->csi_ny);
05498         ALLOC(rt, double,
05499             NOBS);
05500         ALLOC(rz, double,
05501             NOBS);
05502         ALLOC(rlon, double,
05503             NOBS);

```

```

05504     ALLOC(rlat, double,
05505           NOBS);
05506     ALLOC(robs, double,
05507           NOBS);
05508
05509     /* Read observation data... */
05510     read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nobs);
05511
05512     /* Create new file... */
05513     LOG(1, "Write CSI data: %s", filename);
05514     if (! (out = fopen(filename, "w")))
05515         ERRMSG("Cannot create file!");
05516
05517     /* Write header... */
05518     fprintf(out,
05519            "# $1 = time [s]\n"
05520            "# $2 = number of hits (cx)\n"
05521            "# $3 = number of misses (cy)\n"
05522            "# $4 = number of false alarms (cz)\n"
05523            "# $5 = number of observations (cx + cy)\n"
05524            "# $6 = number of forecasts (cx + cz)\n"
05525            "# $7 = bias (ratio of forecasts and observations) [%%]\n"
05526            "# $8 = probability of detection (POD) [%%]\n"
05527            "# $9 = false alarm rate (FAR) [%%]\n"
05528            "# $10 = critical success index (CSI) [%%]\n");
05529     fprintf(out,
05530            "# $11 = hits associated with random chance\n"
05531            "# $12 = equitable threat score (ETS) [%%]\n"
05532            "# $13 = Pearson linear correlation coefficient\n"
05533            "# $14 = Spearman rank-order correlation coefficient\n"
05534            "# $15 = column density mean error (F - O) [kg/m^2]\n"
05535            "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
05536            "# $17 = column density mean absolute error [kg/m^2]\n"
05537            "# $18 = number of data points\n");
05538
05539     /* Set grid box size... */
05540     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
05541     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
05542     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
05543
05544     /* Set horizontal coordinates... */
05545     for (iy = 0; iy < ctl->csi_ny; iy++) {
05546         double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
05547         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
05548     }
05549 }
05550
05551     /* Set time interval... */
05552     double t0 = t - 0.5 * ctl->dt_mod;
05553     double t1 = t + 0.5 * ctl->dt_mod;
05554
05555     /* Allocate... */
05556     ALLOC(modmean, double,
05557           ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05558     ALLOC(obsmean, double,
05559           ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05560     ALLOC(obscount, int,
05561           ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05562
05563     /* Loop over observations... */
05564     for (int i = 0; i < nobs; i++) {
05565
05566         /* Check time... */
05567         if (rt[i] < t0)
05568             continue;
05569         else if (rt[i] >= t1)
05570             break;
05571
05572         /* Check observation data... */
05573         if (!isfinite(robs[i]))
05574             continue;
05575
05576         /* Calculate indices... */
05577         ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
05578         iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
05579         iz = (int) ((rz[i] - ctl->csi_z0) / dz);
05580
05581         /* Check indices... */
05582         if (ix < 0 || ix >= ctl->csi_nx ||
05583             iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05584             continue;
05585
05586         /* Get mean observation index... */
05587         int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05588         obsmean[idx] += robs[i];
05589         obscount[idx]++;
05590     }

```

```

05591
05592 /* Analyze model data... */
05593 for (ip = 0; ip < atm->np; ip++) {
05594
05595     /* Check time... */
05596     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05597         continue;
05598
05599     /* Get indices... */
05600     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
05601     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
05602     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
05603
05604     /* Check indices... */
05605     if (ix < 0 || ix >= ctl->csi_nx ||
05606         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05607         continue;
05608
05609     /* Get total mass in grid cell... */
05610     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05611     modmean[idx] += atm->q[ctl->qnt_m][ip];
05612 }
05613
05614 /* Analyze all grid cells... */
05615 for (ix = 0; ix < ctl->csi_nx; ix++)
05616     for (iy = 0; iy < ctl->csi_ny; iy++)
05617         for (iz = 0; iz < ctl->csi_nz; iz++) {
05618
05619             /* Calculate mean observation index... */
05620             int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05621             if (obscount[idx] > 0)
05622                 obsmean[idx] /= obscount[idx];
05623
05624             /* Calculate column density... */
05625             if (modmean[idx] > 0)
05626                 modmean[idx] /= (1e6 * area[iy]);
05627
05628             /* Calculate CSI... */
05629             if (obscount[idx] > 0) {
05630                 ct++;
05631                 if (obsmean[idx] >= ctl->csi_obsmin &&
05632                     modmean[idx] >= ctl->csi_modmin)
05633                     cx++;
05634                 else if (obsmean[idx] >= ctl->csi_obsmin &&
05635                     modmean[idx] < ctl->csi_modmin)
05636                     cy++;
05637                 else if (obsmean[idx] < ctl->csi_obsmin &&
05638                     modmean[idx] >= ctl->csi_modmin)
05639                     cz++;
05640             }
05641
05642             /* Save data for other verification statistics... */
05643             if (obscount[idx] > 0
05644                 && (obsmean[idx] >= ctl->csi_obsmin
05645                     || modmean[idx] >= ctl->csi_modmin)) {
05646                 x[n] = modmean[idx];
05647                 y[n] = obsmean[idx];
05648                 if (++n > NCSI)
05649                     ERRMSG("Too many data points to calculate statistics!");
05650             }
05651         }
05652
05653 /* Write output... */
05654 if (fmod(t, ctl->csi_dt_out) == 0) {
05655
05656     /* Calculate verification statistics
05657        (https://www.cawcr.gov.au/projects/verification/) ... */
05658     static double work[2 * NCSI];
05659     int n_obs = cx + cy;
05660     int n_for = cx + cz;
05661     double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
05662     double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
05663     double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
05664     double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
05665     double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
05666     double ets = (cx + cy + cz - cx_rd > 0) ?
05667         (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
05668     double rho_p =
05669         (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
05670     double rho_s =
05671         (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
05672     for (int i = 0; i < n; i++)
05673         work[i] = x[i] - y[i];
05674     double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
05675     double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
05676         0.0) : GSL_NAN;
05677     double absdev =

```

```

05678         (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
05679
05680     /* Write... */
05681     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %d\n",
05682            t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
05683            rho_p, rho_s, mean, rmse, absdev, n);
05684
05685     /* Set counters to zero... */
05686     n = ct = cx = cy = cz = 0;
05687 }
05688
05689 /* Free... */
05690 free(modmean);
05691 free(obsmean);
05692 free(obscount);
05693
05694 /* Finalize... */
05695 if (t == ctl->t_stop) {
05696
05697     /* Close output file... */
05698     fclose(out);
05699
05700     /* Free... */
05701     free(area);
05702     free(rt);
05703     free(rz);
05704     free(rlon);
05705     free(rlat);
05706     free(robs);
05707 }
05708 }
05709
05710 /*****
05711 void write_ens(
05712     const char *filename,
05713     ctl_t * ctl,
05714     atm_t * atm,
05715     double t) {
05716
05717     static FILE *out;
05718
05719     static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
05720        x[3], zm[NENS];
05721
05722     static int n[NENS];
05723
05724     /* Set timer... */
05725     SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
05726
05727     /* Check quantities... */
05728     if (ctl->qnt_ens < 0)
05729         ERRMSG("Missing ensemble IDs!");
05730
05731     /* Set time interval... */
05732     double t0 = t - 0.5 * ctl->dt_mod;
05733     double t1 = t + 0.5 * ctl->dt_mod;
05734
05735     /* Init... */
05736     for (int i = 0; i < NENS; i++) {
05737         for (int iq = 0; iq < ctl->nq; iq++)
05738             qm[iq][i] = qs[iq][i] = 0;
05739         xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
05740         n[i] = 0;
05741     }
05742
05743     /* Loop over air parcels... */
05744     for (int ip = 0; ip < atm->np; ip++) {
05745
05746         /* Check time... */
05747         if (atm->time[ip] < t0 || atm->time[ip] > t1)
05748             continue;
05749
05750         /* Check ensemble ID... */
05751         if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
05752             ERRMSG("Ensemble ID is out of range!");
05753
05754         /* Get means... */
05755         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
05756         for (int iq = 0; iq < ctl->nq; iq++) {
05757             qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
05758             qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
05759         }
05760         xm[ctl->qnt_ens][0] += x[0];
05761         xm[ctl->qnt_ens][1] += x[1];
05762         xm[ctl->qnt_ens][2] += x[2];
05763         zm[ctl->qnt_ens] += Z(atm->p[ip]);
05764     }

```

```

05765     n[ctl->qnt_ens]++;
05766 }
05767
05768 /* Create file... */
05769 LOG(1, "Write ensemble data: %s", filename);
05770 if (!(out = fopen(filename, "w")))
05771     ERRMSG("Cannot create file!");
05772
05773 /* Write header... */
05774 fprintf(out,
05775         "# $1 = time [s]\n"
05776         "# $2 = altitude [km]\n"
05777         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05778 for (int iq = 0; iq < ctl->nq; iq++)
05779     fprintf(out, "# %d = %s (mean) [%s]\n", 5 + iq,
05780             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05781 for (int iq = 0; iq < ctl->nq; iq++)
05782     fprintf(out, "# %d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
05783             ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05784 fprintf(out, "# %d = number of members\n\n", 5 + 2 * ctl->nq);
05785
05786 /* Write data... */
05787 for (int i = 0; i < NENS; i++)
05788     if (n[i] > 0) {
05789         cart2geo(xm[i], &dummy, &lon, &lat);
05790         fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
05791         for (int iq = 0; iq < ctl->nq; iq++) {
05792             fprintf(out, " ");
05793             fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
05794         }
05795         for (int iq = 0; iq < ctl->nq; iq++) {
05796             fprintf(out, " ");
05797             double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
05798             fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
05799         }
05800         fprintf(out, " %d\n", n[i]);
05801     }
05802
05803 /* Close file... */
05804 fclose(out);
05805 }
05806
05807 /*****
05808
05809 void write_grid(
05810     const char *filename,
05811     ctl_t *ctl,
05812     met_t *met0,
05813     met_t *met1,
05814     atm_t *atm,
05815     double t) {
05816
05817     double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;
05818
05819     int *ixs, *iys, *izs, *np;
05820
05821     /* Set timer... */
05822     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
05823
05824     /* Write info... */
05825     LOG(1, "Write grid data: %s", filename);
05826
05827     /* Allocate... */
05828     ALLOC(cd, double,
05829           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05830     ALLOC(mass, double,
05831           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05832     ALLOC(vmr_expl, double,
05833           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05834     ALLOC(vmr_impl, double,
05835           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05836     ALLOC(z, double,
05837           ctl->grid_nz);
05838     ALLOC(lon, double,
05839           ctl->grid_nx);
05840     ALLOC(lat, double,
05841           ctl->grid_ny);
05842     ALLOC(area, double,
05843           ctl->grid_ny);
05844     ALLOC(press, double,
05845           ctl->grid_nz);
05846     ALLOC(np, int,
05847           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
05848     ALLOC(ixs, int,
05849           atm->np);
05850     ALLOC(iys, int,
05851           atm->np);

```



```

05852     ALLOC(izs, int,
05853           atm->np);
05854
05855     /* Set grid box size... */
05856     double dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
05857     double dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
05858     double dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
05859
05860     /* Set vertical coordinates... */
05861     #pragma omp parallel for default(shared)
05862     for (int iz = 0; iz < ctl->grid_nz; iz++) {
05863         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
05864         press[iz] = P(z[iz]);
05865     }
05866
05867     /* Set horizontal coordinates... */
05868     for (int ix = 0; ix < ctl->grid_nx; ix++)
05869         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
05870     #pragma omp parallel for default(shared)
05871     for (int iy = 0; iy < ctl->grid_ny; iy++) {
05872         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
05873         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
05874             * cos(lat[iy] * M_PI / 180.);
05875     }
05876
05877     /* Set time interval for output... */
05878     double t0 = t - 0.5 * ctl->dt_mod;
05879     double t1 = t + 0.5 * ctl->dt_mod;
05880
05881     /* Get grid box indices... */
05882     #pragma omp parallel for default(shared)
05883     for (int ip = 0; ip < atm->np; ip++) {
05884         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
05885         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
05886         izs[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05887         if (atm->time[ip] < t0 || atm->time[ip] > t1
05888             || ixs[ip] < 0 || ixs[ip] >= ctl->grid_nx
05889             || iys[ip] < 0 || iys[ip] >= ctl->grid_ny
05890             || izs[ip] < 0 || izs[ip] >= ctl->grid_nz)
05891             izs[ip] = -1;
05892     }
05893
05894     /* Average data... */
05895     for (int ip = 0; ip < atm->np; ip++)
05896         if (izs[ip] >= 0) {
05897             int idx =
05898                 ARRAY_3D(ixs[ip], iys[ip], ctl->grid_ny, izs[ip], ctl->grid_nz);
05899             np[idx]++;
05900             if (ctl->qnt_m >= 0)
05901                 mass[idx] += atm->q[ctl->qnt_m][ip];
05902             if (ctl->qnt_vmr >= 0)
05903                 vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
05904         }
05905
05906     /* Calculate column density and vmr... */
05907     #pragma omp parallel for default(shared)
05908     for (int ix = 0; ix < ctl->grid_nx; ix++)
05909         for (int iy = 0; iy < ctl->grid_ny; iy++)
05910             for (int iz = 0; iz < ctl->grid_nz; iz++) {
05911
05912                 /* Get grid index... */
05913                 int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
05914
05915                 /* Calculate column density... */
05916                 cd[idx] = GSL_NAN;
05917                 if (ctl->qnt_m >= 0)
05918                     cd[idx] = mass[idx] / (1e6 * area[iy]);
05919
05920                 /* Calculate volume mixing ratio (implicit)... */
05921                 vmr_impl[idx] = GSL_NAN;
05922                 if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05923                     vmr_impl[idx] = 0;
05924                     if (mass[idx] > 0) {
05925
05926                         /* Get temperature... */
05927                         double temp;
05928                         INTPOL_INIT;
05929                         intpol_met_time_3d(met0->t, met0->t, met1, met1->t, t, press[iz],
05930                                           lon[ix], lat[iy], &temp, ci, cw, 1);
05931
05932                         /* Calculate volume mixing ratio... */
05933                         vmr_impl[idx] = MA / ctl->molmass * mass[idx]
05934                             / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
05935                     }
05936                 }
05937
05938                 /* Calculate volume mixing ratio (explicit)... */

```

```

05939         if (ctl->qnt_vmr >= 0 && np[idx] > 0)
05940             vmr_expl[idx] /= np[idx];
05941         else
05942             vmr_expl[idx] = GSL_NAN;
05943     }
05944
05945     /* Write ASCII data... */
05946     if (ctl->grid_type == 0)
05947         write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
05948             t, z, lon, lat, area, dz, np);
05949
05950     /* Write netCDF data... */
05951     else if (ctl->grid_type == 1)
05952         write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
05953             t, z, lon, lat, area, dz, np);
05954
05955     /* Error message... */
05956     else
05957         ERRMSG("Grid data format GRID_TYPE unknown!");
05958
05959     /* Free... */
05960     free(cd);
05961     free(mass);
05962     free(vmr_expl);
05963     free(vmr_impl);
05964     free(z);
05965     free(lon);
05966     free(lat);
05967     free(area);
05968     free(press);
05969     free(np);
05970     free(ixs);
05971     free(iys);
05972     free(izs);
05973 }
05974
05975 /*****
05976
05977 void write_grid_asc(
05978     const char *filename,
05979     ctl_t *ctl,
05980     double *cd,
05981     double *vmr_expl,
05982     double *vmr_impl,
05983     double t,
05984     double *z,
05985     double *lon,
05986     double *lat,
05987     double *area,
05988     double dz,
05989     int *np) {
05990
05991     FILE *in, *out;
05992
05993     char line[LEN];
05994
05995     /* Check if gnuplot output is requested... */
05996     if (ctl->grid_gpfile[0] != '-') {
05997
05998         /* Create gnuplot pipe... */
05999         if (!(out = popen("gnuplot", "w")))
06000             ERRMSG("Cannot create pipe to gnuplot!");
06001
06002         /* Set plot filename... */
06003         fprintf(out, "set out \"%s.png\"\\n", filename);
06004
06005         /* Set time string... */
06006         double r;
06007         int year, mon, day, hour, min, sec;
06008         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
06009         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\\n",
06010             year, mon, day, hour, min);
06011
06012         /* Dump gnuplot file to pipe... */
06013         if (!(in = fopen(ctl->grid_gpfile, "r")))
06014             ERRMSG("Cannot open file!");
06015         while (fgets(line, LEN, in))
06016             fprintf(out, "%s", line);
06017         fclose(in);
06018     }
06019
06020     else {
06021
06022         /* Create file... */
06023         if (!(out = fopen(filename, "w")))
06024             ERRMSG("Cannot create file!");
06025     }

```

```

06026
06027 /* Write header... */
06028 fprintf(out,
06029     "# $1 = time [s]\n"
06030     "# $2 = altitude [km]\n"
06031     "# $3 = longitude [deg]\n"
06032     "# $4 = latitude [deg]\n"
06033     "# $5 = surface area [km^2]\n"
06034     "# $6 = layer depth [km]\n"
06035     "# $7 = number of particles [l]\n"
06036     "# $8 = column density (implicit) [kg/m^2]\n"
06037     "# $9 = volume mixing ratio (implicit) [ppv]\n"
06038     "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
06039
06040 /* Write data... */
06041 for (int ix = 0; ix < ctl->grid_nx; ix++) {
06042     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
06043         fprintf(out, "\n");
06044     for (int iy = 0; iy < ctl->grid_ny; iy++) {
06045         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
06046             fprintf(out, "\n");
06047         for (int iz = 0; iz < ctl->grid_nz; iz++) {
06048             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
06049             if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
06050                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
06051                     t, z[iz], lon[ix], lat[iy], area[iy], dz,
06052                     np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
06053         }
06054     }
06055 }
06056
06057 /* Close file... */
06058 fclose(out);
06059 }
06060
06061 /*****
06062
06063 void write_grid_nc(
06064     const char *filename,
06065     ctl_t *ctl,
06066     double *cd,
06067     double *vmr_expl,
06068     double *vmr_impl,
06069     double t,
06070     double *z,
06071     double *lon,
06072     double *lat,
06073     double *area,
06074     double dz,
06075     int *np) {
06076
06077     double *help;
06078
06079     int *help2, ncid, dimid[10], varid;
06080
06081     size_t start[2], count[2];
06082
06083     /* Allocate... */
06084     ALLOC(help, double,
06085         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
06086     ALLOC(help2, int,
06087         ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
06088
06089     /* Create file... */
06090     nc_create(filename, NC_CLOBBER, &ncid);
06091
06092     /* Define dimensions... */
06093     NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
06094     NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[1]));
06095     NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
06096     NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[3]));
06097     NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));
06098
06099     /* Define variables and their attributes... */
06100     NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
06101         "seconds since 2000-01-01 00:00:00 UTC");
06102     NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[1], "altitude", "km");
06103     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
06104     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[3], "longitude", "degrees_east");
06105     NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[1], "layer depth", "km");
06106     NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
06107     NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
06108     NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
06109         "volume mixing ratio (implicit)", "ppv");
06110     NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
06111         "volume mixing ratio (explicit)", "ppv");
06112     NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");

```

```

06113
06114 /* End definitions... */
06115 NC(nc_enddef(ncid));
06116
06117 /* Write data... */
06118 NC_PUT_DOUBLE("time", &t, 0);
06119 NC_PUT_DOUBLE("lon", lon, 0);
06120 NC_PUT_DOUBLE("lat", lat, 0);
06121 NC_PUT_DOUBLE("z", z, 0);
06122 NC_PUT_DOUBLE("area", area, 0);
06123 NC_PUT_DOUBLE("dz", &dz, 0);
06124
06125 for (int ix = 0; ix < ctl->grid_nx; ix++)
06126     for (int iy = 0; iy < ctl->grid_ny; iy++)
06127         for (int iz = 0; iz < ctl->grid_nz; iz++)
06128             help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
06129                 cd[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
06130 NC_PUT_DOUBLE("cd", help, 0);
06131
06132 for (int ix = 0; ix < ctl->grid_nx; ix++)
06133     for (int iy = 0; iy < ctl->grid_ny; iy++)
06134         for (int iz = 0; iz < ctl->grid_nz; iz++)
06135             help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
06136                 vmr_impl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
06137 NC_PUT_DOUBLE("vmr_impl", help, 0);
06138
06139 for (int ix = 0; ix < ctl->grid_nx; ix++)
06140     for (int iy = 0; iy < ctl->grid_ny; iy++)
06141         for (int iz = 0; iz < ctl->grid_nz; iz++)
06142             help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
06143                 vmr_expl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
06144 NC_PUT_DOUBLE("vmr_expl", help, 0);
06145
06146 for (int ix = 0; ix < ctl->grid_nx; ix++)
06147     for (int iy = 0; iy < ctl->grid_ny; iy++)
06148         for (int iz = 0; iz < ctl->grid_nz; iz++)
06149             help2[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
06150                 np[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
06151 NC_PUT_INT("np", help2, 0);
06152
06153 /* Close file... */
06154 NC(nc_close(ncid));
06155
06156 /* Free... */
06157 free(help);
06158 free(help2);
06159 }
06160
06161 /*****
06162
06163 int write_met(
06164     char *filename,
06165     ctl_t *ctl,
06166     met_t *met) {
06167
06168     /* Set timer... */
06169     SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);
06170
06171     /* Write info... */
06172     LOG(1, "Write meteo data: %s", filename);
06173
06174     /* Check compression flags... */
06175     #ifndef ZFP
06176     if (ctl->met_type == 3)
06177         ERRMSG("zfp compression not supported!");
06178     #endif
06179     #ifndef ZSTD
06180     if (ctl->met_type == 4)
06181         ERRMSG("zstd compression not supported!");
06182     #endif
06183
06184     /* Write binary... */
06185     if (ctl->met_type >= 1 && ctl->met_type <= 4) {
06186
06187         /* Create file... */
06188         FILE *out;
06189         if (!(out = fopen(filename, "w")))
06190             ERRMSG("Cannot create file!");
06191
06192         /* Write type of binary data... */
06193         FWRITE(&ctl->met_type, int,
06194             1,
06195             out);
06196
06197         /* Write version of binary data... */
06198         int version = 100;
06199         FWRITE(&version, int,

```

```

06200         1,
06201         out);
06202
06203     /* Write grid data... */
06204     FWRITE(&met->time, double,
06205           1,
06206           out);
06207     FWRITE(&met->nx, int,
06208           1,
06209           out);
06210     FWRITE(&met->ny, int,
06211           1,
06212           out);
06213     FWRITE(&met->np, int,
06214           1,
06215           out);
06216     FWRITE(met->lon, double,
06217           (size_t) met->nx,
06218           out);
06219     FWRITE(met->lat, double,
06220           (size_t) met->ny,
06221           out);
06222     FWRITE(met->p, double,
06223           (size_t) met->np,
06224           out);
06225
06226     /* Write surface data... */
06227     write_met_bin_2d(out, met, met->ps, "PS");
06228     write_met_bin_2d(out, met, met->ts, "TS");
06229     write_met_bin_2d(out, met, met->zs, "ZS");
06230     write_met_bin_2d(out, met, met->us, "US");
06231     write_met_bin_2d(out, met, met->vs, "VS");
06232     write_met_bin_2d(out, met, met->pbl, "PBL");
06233     write_met_bin_2d(out, met, met->pt, "PT");
06234     write_met_bin_2d(out, met, met->tt, "TT");
06235     write_met_bin_2d(out, met, met->zt, "ZT");
06236     write_met_bin_2d(out, met, met->h2ot, "H2OT");
06237     write_met_bin_2d(out, met, met->pct, "PCT");
06238     write_met_bin_2d(out, met, met->pcb, "PCB");
06239     write_met_bin_2d(out, met, met->cl, "CL");
06240     write_met_bin_2d(out, met, met->plcl, "PLCL");
06241     write_met_bin_2d(out, met, met->plfc, "PLFC");
06242     write_met_bin_2d(out, met, met->pel, "PEL");
06243     write_met_bin_2d(out, met, met->cape, "CAPE");
06244     write_met_bin_2d(out, met, met->cin, "CIN");
06245
06246     /* Write level data... */
06247     write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
06248     write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
06249     write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
06250     write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
06251     write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
06252     write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
06253     write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
06254     write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
06255     write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
06256     write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);
06257
06258     /* Write final flag... */
06259     int final = 999;
06260     FWRITE(&final, int,
06261           1,
06262           out);
06263
06264     /* Close file... */
06265     fclose(out);
06266 }
06267
06268 return 0;
06269 }
06270
06271 /*****
06272
06273 void write_met_bin_2d(
06274     FILE * out,
06275     met_t * met,
06276     float var[EX][EY],
06277     char *varname) {
06278
06279     float *help;
06280
06281     /* Allocate... */
06282     ALLOC(help, float,
06283           EX * EY);
06284
06285     /* Copy data... */
06286     for (int ix = 0; ix < met->nx; ix++)

```

```

06287     for (int iy = 0; iy < met->ny; iy++)
06288         help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];
06289
06290     /* Write uncompressed data... */
06291     LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
06292     FWRITE(help, float,
06293         (size_t) (met->nx * met->ny),
06294         out);
06295
06296     /* Free... */
06297     free(help);
06298 }
06299
06300 /*****
06301
06302 void write_met_bin_3d(
06303     FILE * out,
06304     ctl_t * ctl,
06305     met_t * met,
06306     float var[EX][EY][EP],
06307     char *varname,
06308     int precision,
06309     double tolerance) {
06310
06311     float *help;
06312
06313     /* Allocate... */
06314     ALLOC(help, float,
06315         EX * EY * EP);
06316
06317     /* Copy data... */
06318     #pragma omp parallel for default(shared) collapse(2)
06319     for (int ix = 0; ix < met->nx; ix++)
06320         for (int iy = 0; iy < met->ny; iy++)
06321             for (int ip = 0; ip < met->np; ip++)
06322                 help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];
06323
06324     /* Write uncompressed data... */
06325     if (ctl->met_type == 1) {
06326         LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
06327         FWRITE(help, float,
06328             (size_t) (met->nx * met->ny * met->np),
06329             out);
06330     }
06331
06332     /* Write packed data... */
06333     else if (ctl->met_type == 2)
06334         compress_pack(varname, help, (size_t) (met->ny * met->nx),
06335             (size_t) met->np, 0, out);
06336
06337     /* Write zfp data... */
06338     #ifdef ZFP
06339     else if (ctl->met_type == 3)
06340         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
06341             tolerance, 0, out);
06342     #endif
06343
06344     /* Write zstd data... */
06345     #ifdef ZSTD
06346     else if (ctl->met_type == 4)
06347         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
06348             out);
06349     #endif
06350
06351     /* Unknown method... */
06352     else {
06353         ERRMSG("MET_TYPE not supported!");
06354         LOG(3, "%d %g", precision, tolerance);
06355     }
06356
06357     /* Free... */
06358     free(help);
06359 }
06360
06361 /*****
06362
06363 void write_prof(
06364     const char *filename,
06365     ctl_t * ctl,
06366     met_t * met0,
06367     met_t * met1,
06368     atm_t * atm,
06369     double t) {
06370
06371     static FILE *out;
06372
06373     static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,

```

```

06374     dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;
06375
06376     static int nob, *obscount, ip, okay;
06377
06378     /* Set timer... */
06379     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
06380
06381     /* Init... */
06382     if (t == ctl->t_start) {
06383
06384         /* Check quantity index for mass... */
06385         if (ctl->qnt_m < 0)
06386             ERRMSG("Need quantity mass!");
06387
06388         /* Check molar mass... */
06389         if (ctl->molmass <= 0)
06390             ERRMSG("Specify molar mass!");
06391
06392         /* Allocate... */
06393         ALLOC(lon, double,
06394              ctl->prof_nx);
06395         ALLOC(lat, double,
06396              ctl->prof_ny);
06397         ALLOC(area, double,
06398              ctl->prof_ny);
06399         ALLOC(z, double,
06400              ctl->prof_nz);
06401         ALLOC(press, double,
06402              ctl->prof_nz);
06403         ALLOC(rt, double,
06404              NOBS);
06405         ALLOC(rz, double,
06406              NOBS);
06407         ALLOC(rlon, double,
06408              NOBS);
06409         ALLOC(rlat, double,
06410              NOBS);
06411         ALLOC(robs, double,
06412              NOBS);
06413
06414         /* Read observation data... */
06415         read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nob);
06416
06417         /* Create new output file... */
06418         LOG(1, "Write profile data: %s", filename);
06419         if (!(out = fopen(filename, "w")))
06420             ERRMSG("Cannot create file!");
06421
06422         /* Write header... */
06423         fprintf(out,
06424              "# $1 = time [s]\n"
06425              "# $2 = altitude [km]\n"
06426              "# $3 = longitude [deg]\n"
06427              "# $4 = latitude [deg]\n"
06428              "# $5 = pressure [hPa]\n"
06429              "# $6 = temperature [K]\n"
06430              "# $7 = volume mixing ratio [ppv]\n"
06431              "# $8 = H2O volume mixing ratio [ppv]\n"
06432              "# $9 = O3 volume mixing ratio [ppv]\n"
06433              "# $10 = observed BT index [K]\n"
06434              "# $11 = number of observations\n");
06435
06436         /* Set grid box size... */
06437         dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
06438         dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
06439         dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
06440
06441         /* Set vertical coordinates... */
06442         for (int iz = 0; iz < ctl->prof_nz; iz++) {
06443             z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
06444             press[iz] = P(z[iz]);
06445         }
06446
06447         /* Set horizontal coordinates... */
06448         for (int ix = 0; ix < ctl->prof_nx; ix++)
06449             lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
06450         for (int iy = 0; iy < ctl->prof_ny; iy++) {
06451             lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
06452             area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
06453                 * cos(lat[iy] * M_PI / 180.);
06454         }
06455     }
06456
06457     /* Set time interval... */
06458     double t0 = t - 0.5 * ctl->dt_mod;
06459     double t1 = t + 0.5 * ctl->dt_mod;
06460

```

```

06461  /* Allocate... */
06462  ALLOC(mass, double,
06463        ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
06464  ALLOC(obsmean, double,
06465        ctl->prof_nx * ctl->prof_ny);
06466  ALLOC(obscount, int,
06467        ctl->prof_nx * ctl->prof_ny);
06468
06469  /* Loop over observations... */
06470  for (int i = 0; i < nobs; i++) {
06471
06472      /* Check time... */
06473      if (rt[i] < t0)
06474          continue;
06475      else if (rt[i] >= t1)
06476          break;
06477
06478      /* Check observation data... */
06479      if (!isfinite(robs[i]))
06480          continue;
06481
06482      /* Calculate indices... */
06483      int ix = (int) ((rlon[i] - ctl->prof_lon0) / dlon);
06484      int iy = (int) ((rlat[i] - ctl->prof_lat0) / dlat);
06485
06486      /* Check indices... */
06487      if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
06488          continue;
06489
06490      /* Get mean observation index... */
06491      int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
06492      obsmean[idx] += robs[i];
06493      obscount[idx]++;
06494  }
06495
06496  /* Analyze model data... */
06497  for (ip = 0; ip < atm->np; ip++) {
06498
06499      /* Check time... */
06500      if (atm->time[ip] < t0 || atm->time[ip] > t1)
06501          continue;
06502
06503      /* Get indices... */
06504      int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
06505      int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
06506      int iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
06507
06508      /* Check indices... */
06509      if (ix < 0 || ix >= ctl->prof_nx ||
06510          iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
06511          continue;
06512
06513      /* Get total mass in grid cell... */
06514      int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06515      mass[idx] += atm->q[ctl->qnt_m][ip];
06516  }
06517
06518  /* Extract profiles... */
06519  for (int ix = 0; ix < ctl->prof_nx; ix++)
06520      for (int iy = 0; iy < ctl->prof_ny; iy++) {
06521          int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
06522          if (obscount[idx2] > 0) {
06523
06524              /* Check profile... */
06525              okay = 0;
06526              for (int iz = 0; iz < ctl->prof_nz; iz++) {
06527                  int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06528                  if (mass[idx3] > 0) {
06529                      okay = 1;
06530                      break;
06531                  }
06532              }
06533              if (!okay)
06534                  continue;
06535
06536              /* Write output... */
06537              fprintf(out, "\n");
06538
06539              /* Loop over altitudes... */
06540              for (int iz = 0; iz < ctl->prof_nz; iz++) {
06541
06542                  /* Get temperature, water vapor, and ozone... */
06543                  INTPOL_INIT;
06544                  intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
06545                                     lon[ix], lat[iy], &temp, ci, cw, 1);
06546                  intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
06547                                     lon[ix], lat[iy], &h2o, ci, cw, 0);

```



```

06548         intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
06549                             lon[ix], lat[iy], &o3, ci, cw, 0);
06550
06551         /* Calculate volume mixing ratio... */
06552         int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06553         vmr = MA / ctl->molmass * mass[idx3]
06554             / (RHO(press[iz], temp) * area[iy] * dz * 1e9);
06555
06556         /* Write output... */
06557         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
06558                 t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
06559                 obsmean[idx2] / obscount[idx2], obscount[idx2]);
06560     }
06561 }
06562 }
06563
06564 /* Free... */
06565 free(mass);
06566 free(obsmean);
06567 free(obscount);
06568
06569 /* Finalize... */
06570 if (t == ctl->t_stop) {
06571
06572     /* Close output file... */
06573     fclose(out);
06574
06575     /* Free... */
06576     free(lon);
06577     free(lat);
06578     free(area);
06579     free(z);
06580     free(press);
06581     free(rt);
06582     free(rz);
06583     free(rlon);
06584     free(rlat);
06585     free(robs);
06586 }
06587 }
06588
06589 /*****
06590
06591 void write_sample(
06592     const char *filename,
06593     ctl_t * ctl,
06594     met_t * met0,
06595     met_t * met1,
06596     atm_t * atm,
06597     double t) {
06598
06599     static FILE *out;
06600
06601     static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;
06602
06603     static int nobs;
06604
06605     /* Set timer... */
06606     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
06607
06608     /* Init... */
06609     if (t == ctl->t_start) {
06610
06611         /* Allocate... */
06612         ALLOC(rt, double,
06613              NOBS);
06614         ALLOC(rz, double,
06615              NOBS);
06616         ALLOC(rlon, double,
06617              NOBS);
06618         ALLOC(rlat, double,
06619              NOBS);
06620         ALLOC(robs, double,
06621              NOBS);
06622
06623         /* Read observation data... */
06624         read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06625
06626         /* Create output file... */
06627         LOG(1, "Write sample data: %s", filename);
06628         if (!(out = fopen(filename, "w")))
06629             ERRMSG("Cannot create file!");
06630
06631         /* Write header... */
06632         fprintf(out,
06633                 "# $1 = time [s]\n"
06634                 "# $2 = altitude [km]\n"

```

```

06635         "# $3 = longitude [deg]\n"
06636         "# $4 = latitude [deg]\n"
06637         "# $5 = surface area [km^2]\n"
06638         "# $6 = layer depth [km]\n"
06639         "# $7 = number of particles [1]\n"
06640         "# $8 = column density [kg/m^2]\n"
06641         "# $9 = volume mixing ratio [ppv]\n"
06642         "# $10 = observed BT index [K]\n\n");
06643
06644     /* Set latitude range, squared radius, and area... */
06645     dlat = DY2DEG(ctl->sample_dx);
06646     rmax2 = SQR(ctl->sample_dx);
06647     area = M_PI * rmax2;
06648 }
06649
06650 /* Set time interval for output... */
06651 double t0 = t - 0.5 * ctl->dt_mod;
06652 double t1 = t + 0.5 * ctl->dt_mod;
06653
06654 /* Loop over observations... */
06655 for (int i = 0; i < nob; i++) {
06656
06657     /* Check time... */
06658     if (rt[i] < t0)
06659         continue;
06660     else if (rt[i] >= t1)
06661         break;
06662
06663     /* Calculate Cartesian coordinates... */
06664     double x0[3];
06665     geo2cart(0, rlon[i], rlat[i], x0);
06666
06667     /* Set pressure range... */
06668     double rp = P(rz[i]);
06669     double ptop = P(rz[i] + ctl->sample_dz);
06670     double pbot = P(rz[i] - ctl->sample_dz);
06671
06672     /* Init... */
06673     double mass = 0;
06674     int np = 0;
06675
06676     /* Loop over air parcels... */
06677 #pragma omp parallel for default(shared) reduction(+:mass,np)
06678     for (int ip = 0; ip < atm->np; ip++) {
06679
06680         /* Check time... */
06681         if (atm->time[ip] < t0 || atm->time[ip] > t1)
06682             continue;
06683
06684         /* Check latitude... */
06685         if (fabs(rlat[i] - atm->lat[ip]) > dlat)
06686             continue;
06687
06688         /* Check horizontal distance... */
06689         double x1[3];
06690         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06691         if (DIST2(x0, x1) > rmax2)
06692             continue;
06693
06694         /* Check pressure... */
06695         if (ctl->sample_dz > 0)
06696             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
06697                 continue;
06698
06699         /* Add mass... */
0700         if (ctl->qnt_m >= 0)
0701             mass += atm->q[ctl->qnt_m][ip];
0702         np++;
0703     }
0704
0705     /* Calculate column density... */
0706     double cd = mass / (1e6 * area);
0707
0708     /* Calculate volume mixing ratio... */
0709     double vmr = 0;
0710     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
0711         if (mass > 0) {
0712
0713             /* Get temperature... */
0714             double temp;
0715             INTPOL_INIT;
0716             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
0717                               rlon[i], rlat[i], &temp, ci, cw, 1);
0718
0719             /* Calculate volume mixing ratio... */
0720             vmr = MA / ctl->molmass * mass
0721                 / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);

```

```

06722     }
06723     } else
06724     {
06725         vmr = GSL_NAN;
06726
06727         /* Write output... */
06728         fprintf(out, "%.2f %g %g %g %g %g %g %g\n", rt[i], rz[i],
06729             rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
06730     }
06731
06732     /* Finalize..... */
06733     if (t == ctl->t_stop) {
06734
06735         /* Close output file... */
06736         fclose(out);
06737
06738         /* Free... */
06739         free(rt);
06740         free(rz);
06741         free(rlon);
06742         free(rlat);
06743         free(robs);
06744     }
06745 }
06746 /*****
06747 void write_station(
06748     const char *filename,
06749     ctl_t * ctl,
06750     atm_t * atm,
06751     double t) {
06752
06753     static FILE *out;
06754
06755     static double rmax2, x0[3], x1[3];
06756
06757     /* Set timer... */
06758     SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
06759
06760     /* Init... */
06761     if (t == ctl->t_start) {
06762
06763         /* Write info... */
06764         LOG(1, "Write station data: %s", filename);
06765
06766         /* Create new file... */
06767         if (!(out = fopen(filename, "w")))
06768             ERRMSG("Cannot create file!");
06769
06770         /* Write header... */
06771         fprintf(out,
06772             "# $1 = time [s]\n"
06773             "# $2 = altitude [km]\n"
06774             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
06775         for (int iq = 0; iq < ctl->nq; iq++)
06776             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
06777                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
06778         fprintf(out, "\n");
06779
06780         /* Set geolocation and search radius... */
06781         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
06782         rmax2 = SQR(ctl->stat_r);
06783     }
06784
06785     /* Set time interval for output... */
06786     double t0 = t - 0.5 * ctl->dt_mod;
06787     double t1 = t + 0.5 * ctl->dt_mod;
06788
06789     /* Loop over air parcels... */
06790     for (int ip = 0; ip < atm->np; ip++) {
06791
06792         /* Check time... */
06793         if (atm->time[ip] < t0 || atm->time[ip] > t1)
06794             continue;
06795
06796         /* Check time range for station output... */
06797         if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
06798             continue;
06799
06800         /* Check station flag... */
06801         if (ctl->qnt_stat >= 0)
06802             if (atm->q[ctl->qnt_stat][ip])
06803                 continue;
06804
06805         /* Get Cartesian coordinates... */
06806         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06807     }
06808 }

```

```

06809      /* Check horizontal distance... */
06810      if (DIST2(x0, x1) > rmax2)
06811          continue;
06812
06813      /* Set station flag... */
06814      if (ctl->qnt_stat >= 0)
06815          atm->q[ctl->qnt_stat][ip] = 1;
06816
06817      /* Write data... */
06818      fprintf(out, "%.2f %g %g %g",
06819              atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
06820      for (int iq = 0; iq < ctl->nq; iq++) {
06821          fprintf(out, " ");
06822          fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
06823      }
06824      fprintf(out, "\n");
06825  }
06826
06827      /* Close file... */
06828      if (t == ctl->t_stop)
06829          fclose(out);
06830  }

```

5.23 libtrac.h File Reference

MPTRAC library declarations.

```

#include <ctype.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_statistics.h>
#include <math.h>
#include <netcdf.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

```

Data Structures

- struct [ctl_t](#)
Control parameters.
- struct [atm_t](#)
Atmospheric data.
- struct [cache_t](#)
Cache data.
- struct [clim_t](#)
Climatological data.
- struct [met_t](#)
Meteo data.

Macros

- #define **CPD** 1003.5
Specific heat of dry air at constant pressure [J/(kg K)].
- #define **EPS** (**MH2O** / **MA**)
Ratio of the specific gas constant of dry air and water vapor [1].
- #define **G0** 9.80665
Standard gravity [m/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 **NCSI** 1000000
Maximum number of data points for CSI calculation.
- #define **EP** 140
Maximum number of pressure levels for meteo data.
- #define **EX** 1201
Maximum number of longitudes for meteo data.
- #define **EY** 601
Maximum number of latitudes for meteo data.
- #define **NENS** 2000
Maximum number of data points for ensemble analysis.
- #define **NOBS** 10000000
Maximum number of observation data points.
- #define **NTHREADS** 512
Maximum number of OpenMP threads.
- #define **CY** 250

- Maximum number of latitudes for climatological data.*

 - #define CP 60
- Maximum number of pressure levels for climatological data.*

 - #define CT 12
- Maximum number of time steps for climatological data.*

 - #define ALLOC(ptr, type, n)

Allocate and clear memory.
- Get 2-D array index.*

 - #define ARRAY_2D(ix, iy, ny) ((ix) * (ny) + (iy))
- Get 3-D array index.*

 - #define ARRAY_3D(ix, iy, ny, iz, nz) (((ix)*(ny) + (iy)) * (nz) + (iz))
- Convert degrees to zonal distance.*

 - #define DEG2DX(dlon, lat) ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
- Convert degrees to meridional distance.*

 - #define DEG2DY(dlat) ((dlat) * M_PI * RE / 180.)
- Convert pressure change to vertical distance.*

 - #define DP2DZ(dp, p) (- (dp) * H0 / (p))
- Convert zonal distance to degrees.*

 - #define DX2DEG(dx, lat)
- Convert meridional distance to degrees.*

 - #define DY2DEG(dy) ((dy) * 180. / (M_PI * RE))
- Convert vertical distance to pressure change.*

 - #define DZ2DP(dz, p) (- (dz) * (p) / H0)
- Compute Cartesian distance between two vectors.*

 - #define DIST(a, b) sqrt(DIST2(a, b))
- Compute squared 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 dot product of two vectors.*

 - #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
- Compute floating point modulo.*

 - #define FMOD(x, y) ((x) - (int) ((x) / (y)) * (y))
- Read binary data.*

 - #define FREAD(ptr, type, size, out)
- Write binary data.*

 - #define FWRITE(ptr, type, size, out)
- Initialize cache variables for interpolation.*

 - #define INTPOL_INIT double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};
- 2-D interpolation of a meteo variable.*

 - #define INTPOL_2D(var, init)
- 3-D interpolation of a meteo variable.*

 - #define INTPOL_3D(var, init)
- Spatial interpolation of all meteo data.*

 - #define INTPOL_SPACE_ALL(p, lon, lat)
- Temporal interpolation of all meteo data.*

 - #define INTPOL_TIME_ALL(time, p, lon, lat)
- Calculate lapse rate between pressure levels.*

 - #define LAPSE(p1, t1, p2, t2)
- Compute linear interpolation.*

 - #define LIN(x0, y0, x1, y1, x) ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
- Execute netCDF library command and check result.*

 - #define NC(cmd)

- #define **NC_DEF_VAR**(varname, type, ndims, dims, long_name, units)
Define netCDF variable.
- #define **NC_GET_DOUBLE**(varname, ptr, force)
Read netCDF double array.
- #define **NC_INQ_DIM**(dimname, ptr, min, max)
Read netCDF dimension.
- #define **NC_PUT_DOUBLE**(varname, ptr, hyperslab)
Write netCDF double array.
- #define **NC_PUT_INT**(varname, ptr, hyperslab)
Write netCDF integer array.
- #define **NC_PUT_ATT**(varname, attname, text)
Set netCDF attribute.
- #define **NC_PUT_ATT_GLOBAL**(attname, text) **NC**(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));
Set netCDF global attribute.
- #define **NC_PUT_FLOAT**(varname, ptr, hyperslab)
Write netCDF float array.
- #define **NN**(x0, y0, x1, y1, x) (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))
Compute nearest neighbor interpolation.
- #define **NORM**(a) sqrt(**DOTP**(a, a))
Compute norm of a vector.
- #define **P**(z) (**P0** * exp(-(z) / **H0**))
Convert altitude to pressure.
- #define **PSAT**(t) (6.112 * exp(17.62 * ((t) - **T0**) / (243.12 + (t) - **T0**)))
Compute saturation pressure over water (WMO, 2018).
- #define **PSICE**(t) (6.112 * exp(22.46 * ((t) - **T0**) / (272.62 + (t) - **T0**)))
Compute saturation pressure over ice (WMO, 2018).
- #define **PW**(p, h2o)
Calculate partial water vapor pressure.
- #define **RH**(p, t, h2o) (**PW**(p, h2o) / **PSAT**(t) * 100.)
Compute relative humidity over water.
- #define **RHICE**(p, t, h2o) (**PW**(p, h2o) / **PSICE**(t) * 100.)
Compute relative humidity over ice.
- #define **RHO**(p, t) (100. * (p) / (**RA** * (t)))
Compute density of air.
- #define **SET_ATM**(qnt, val)
Set atmospheric quantity value.
- #define **SET_QNT**(qnt, name, longname, unit)
Set atmospheric quantity index.
- #define **SH**(h2o) (**EPS** * **GSL_MAX**((h2o), 0.1e-6))
Compute specific humidity from water vapor volume mixing ratio.
- #define **SQR**(x) ((x)*(x))
Compute square.
- #define **SWAP**(x, y, type) do {type tmp = x; x = y; y = tmp;} while(0);
Swap macro.
- #define **TDEW**(p, h2o)
Calculate dew point temperature (WMO, 2018).
- #define **TICE**(p, h2o)
Calculate frost point temperature (WMO, 2018).
- #define **THETA**(p, t) ((t) * pow(1000. / (p), 0.286))
Compute potential temperature.

- #define **THETA**(p, t, h2o) (**TVIRT**(**THETA**((p), (t)), **GSL_MAX**((h2o), 0.1e-6)))
Compute virtual potential temperature.
- #define **TOK**(line, tok, format, var)
Get string tokens.
- #define **TVIRT**(t, h2o) ((t) * (1. + (1. - **EPS**) * **GSL_MAX**((h2o), 0.1e-6)))
Compute virtual temperature.
- #define **Z**(p) (**H0** * **log**(**P0** / (p)))
Convert pressure to altitude.
- #define **ZDIFF**(lnp0, t0, h2o0, lnp1, t1, h2o1)
Calculate geopotential height difference.
- #define **ZETA**(ps, p, t)
Calculate zeta vertical coordinate.
- #define **LOGLEV** 2
Level of log messages (0=none, 1=basic, 2=detailed, 3=debug).
- #define **LOG**(level, ...)
Print log message.
- #define **WARN**(...)
Print warning message.
- #define **ERRMSG**(...)
Print error message and quit program.
- #define **PRINT**(format, var)
Print macro for debugging.
- #define **NTIMER** 100
Maximum number of timers.
- #define **PRINT_TIMERS** **timer**("END", "END", 1);
Print timers.
- #define **SELECT_TIMER**(id, group, color)
Select timer.
- #define **START_TIMERS** **NVTX_PUSH**("START", **NVTX_CPU**);
Start timers.
- #define **STOP_TIMERS** **NVTX_POP**;
Stop timers.
- #define **NVTX_PUSH**(range_title, range_color) {}
- #define **NVTX_POP** {}

Functions

- void **thrustSortWrapper** (double *__restrict__ c, int n, int *__restrict__ index)
Wrapper to Thrust sorting function.
- double **buoyancy_frequency** (double p0, double t0, double p1, double t1)
Calculate buoyancy frequency.
- void **cart2geo** (double *x, double *z, double *lon, double *lat)
Convert Cartesian coordinates to geolocation.
- int **check_finite** (const double x)
Check if x is finite.
- double **clim_hno3** (**clim_t** *clim, double t, double lat, double p)
Climatology of HNO3 volume mixing ratios.
- void **clim_hno3_init** (**clim_t** *clim)
Initialization function for HNO3 climatology.
- double **clim_oh** (**clim_t** *clim, double t, double lat, double p)

- Climatology of OH number concentrations.*
- double `clim_oh_diurnal` (`ctl_t` *ctl, `clim_t` *clim, double t, double p, double lon, double lat)
- Climatology of OH number concentrations with diurnal variation.*
- void `clim_oh_init` (`ctl_t` *ctl, `clim_t` *clim)
- Initialization function for OH climatology.*
- double `clim_oh_init_help` (double beta, double time, double lat)
- Apply diurnal correction to OH climatology.*
- double `clim_h2o2` (`clim_t` *clim, double t, double lat, double p)
- Climatology of H2O2 number concentrations.*
- void `clim_h2o2_init` (`ctl_t` *ctl, `clim_t` *clim)
- Initialization function for H2O2 climatology.*
- double `clim_tropo` (`clim_t` *clim, double t, double lat)
- Climatology of tropopause pressure.*
- void `clim_tropo_init` (`clim_t` *clim)
- Initialize tropopause climatology.*
- void `compress_pack` (char *varname, float *array, size_t nxy, size_t nz, int decompress, FILE *inout)
- Pack or unpack array.*
- void `day2doy` (int year, int mon, int day, int *doy)
- Compress or decompress array with zfp.*
- void `doy2day` (int year, int doy, int *mon, int *day)
- Get date from day of year.*
- void `geo2cart` (double z, double lon, double lat, double *x)
- Convert geolocation to Cartesian coordinates.*
- void `get_met` (`ctl_t` *ctl, `clim_t` *clim, double t, `met_t` **met0, `met_t` **met1)
- Get meteo data for given time step.*
- void `get_met_help` (`ctl_t` *ctl, double t, int direct, char *metbase, double dt_met, char *filename)
- Get meteo data for time step.*
- void `get_met_replace` (char *orig, char *search, char *repl)
- Replace template strings in filename.*
- void `intpol_met_space_3d` (`met_t` *met, float array[EX][EY][EP], double p, double lon, double lat, double *var, int *ci, double *cw, int init)
- Spatial interpolation of meteo data.*
- void `intpol_met_space_2d` (`met_t` *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)
- Spatial interpolation of meteo data.*
- void `intpol_met_time_3d` (`met_t` *met0, float array0[EX][EY][EP], `met_t` *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)
- Spatial interpolation of meteo data.*
- void `intpol_met_time_2d` (`met_t` *met0, float array0[EX][EY], `met_t` *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)
- Temporal interpolation of meteo data.*
- void `jsec2time` (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)
- Temporal interpolation of meteo data.*
- double `lapse_rate` (double t, double h2o)
- Calculate moist adiabatic lapse rate.*
- int `locate_irr` (double *xx, int n, double x)
- Find array index for irregular grid.*
- int `locate_reg` (double *xx, int n, double x)
- Find array index for regular grid.*
- double `nat_temperature` (double p, double h2o, double hno3)
- Calculate NAT existence temperature.*

- void `quicksort` (double arr[], int brr[], int low, int high)
Parallel quicksort.
- int `quicksort_partition` (double arr[], int brr[], int low, int high)
Partition function for quicksort.
- int `read_atm` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
Read atmospheric data.
- int `read_atm_asc` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
Read atmospheric data in ASCII format.
- int `read_atm_bin` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
Read atmospheric data in binary format.
- int `read_atm_clams` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
Read atmospheric data in CLaMS format.
- int `read_atm_nc` (const char *filename, `ctl_t` *ctl, `atm_t` *atm)
Read atmospheric data in netCDF format.
- void `read_clim` (`ctl_t` *ctl, `clim_t` *clim)
Read climatological data.
- void `read_ctl` (const char *filename, int argc, char *argv[], `ctl_t` *ctl)
Read control parameters.
- int `read_met` (char *filename, `ctl_t` *ctl, `clim_t` *clim, `met_t` *met)
Read meteo data file.
- void `read_met_bin_2d` (FILE *out, `met_t` *met, float var[EX][EY], char *varname)
Read 2-D meteo variable.
- void `read_met_bin_3d` (FILE *in, `ctl_t` *ctl, `met_t` *met, float var[EX][EY][EP], char *varname, int precision, double tolerance)
Read 3-D meteo variable.
- void `read_met_cape` (`clim_t` *clim, `met_t` *met)
Calculate convective available potential energy.
- void `read_met_cloud` (`ctl_t` *ctl, `met_t` *met)
Calculate cloud properties.
- void `read_met_detrend` (`ctl_t` *ctl, `met_t` *met)
Apply detrending method to temperature and winds.
- void `read_met_extrapolate` (`met_t` *met)
Extrapolate meteo data at lower boundary.
- void `read_met_geopot` (`ctl_t` *ctl, `met_t` *met)
Calculate geopotential heights.
- void `read_met_grid` (char *filename, int ncid, `ctl_t` *ctl, `met_t` *met)
Read coordinates of meteo data.
- void `read_met_levels` (int ncid, `ctl_t` *ctl, `met_t` *met)
Read meteo data on vertical levels.
- void `read_met_ml2pl` (`ctl_t` *ctl, `met_t` *met, float var[EX][EY][EP])
Convert meteo data from model levels to pressure levels.
- int `read_met_nc_2d` (int ncid, char *varname, char *varname2, `ctl_t` *ctl, `met_t` *met, float dest[EX][EY], float scl, int init)
Read and convert 2D variable from meteo data file.
- int `read_met_nc_3d` (int ncid, char *varname, char *varname2, `ctl_t` *ctl, `met_t` *met, float dest[EX][EY][EP], float scl, int init)
Read and convert 3D variable from meteo data file.
- void `read_met_pbl` (`met_t` *met)
Calculate pressure of the boundary layer.
- void `read_met_periodic` (`met_t` *met)
Create meteo data with periodic boundary conditions.

- void `read_met_pv` (`met_t *met`)
Calculate potential vorticity.
- void `read_met_sample` (`ctl_t *ctl`, `met_t *met`)
Downsampling of meteo data.
- void `read_met_surface` (`int ncid`, `met_t *met`, `ctl_t *ctl`)
Read surface data.
- void `read_met_tropo` (`ctl_t *ctl`, `clim_t *clim`, `met_t *met`)
Calculate tropopause data.
- void `read_obs` (`char *filename`, `double *rt`, `double *rz`, `double *rlon`, `double *rlat`, `double *robs`, `int *nobs`)
Read observation data.
- double `scan_ctl` (`const char *filename`, `int argc`, `char *argv[]`, `const char *varname`, `int arridx`, `const char *defvalue`, `char *value`)
Read a control parameter from file or command line.
- double `sedi` (`double p`, `double T`, `double rp`, `double rhop`)
Calculate sedimentation velocity.
- void `spline` (`double *x`, `double *y`, `int n`, `double *x2`, `double *y2`, `int n2`, `int method`)
Spline interpolation.
- float `stddev` (`float *data`, `int n`)
Calculate standard deviation.
- double `sza` (`double sec`, `double lon`, `double lat`)
Calculate solar zenith angle.
- void `time2jsec` (`int year`, `int mon`, `int day`, `int hour`, `int min`, `int sec`, `double remain`, `double *jsec`)
Convert date to seconds.
- void `timer` (`const char *name`, `const char *group`, `int output`)
Measure wall-clock time.
- double `tropo_weight` (`clim_t *clim`, `double t`, `double lat`, `double p`)
Get weighting factor based on tropopause distance.
- void `write_atm` (`const char *filename`, `ctl_t *ctl`, `atm_t *atm`, `double t`)
Write atmospheric data.
- void `write_atm_asc` (`const char *filename`, `ctl_t *ctl`, `atm_t *atm`, `double t`)
Write atmospheric data in ASCII format.
- void `write_atm_bin` (`const char *filename`, `ctl_t *ctl`, `atm_t *atm`)
Write atmospheric data in binary format.
- void `write_atm_clams` (`ctl_t *ctl`, `atm_t *atm`, `double t`)
Write atmospheric data in CLaMS format.
- void `write_atm_nc` (`const char *filename`, `ctl_t *ctl`, `atm_t *atm`)
Write atmospheric data in netCDF format.
- void `write_csi` (`const char *filename`, `ctl_t *ctl`, `atm_t *atm`, `double t`)
Write CSI data.
- void `write_ens` (`const char *filename`, `ctl_t *ctl`, `atm_t *atm`, `double t`)
Write ensemble data.
- void `write_grid` (`const char *filename`, `ctl_t *ctl`, `met_t *met0`, `met_t *met1`, `atm_t *atm`, `double t`)
Write gridded data.
- void `write_grid_asc` (`const char *filename`, `ctl_t *ctl`, `double *cd`, `double *vmr_expl`, `double *vmr_impl`, `double t`, `double *z`, `double *lon`, `double *lat`, `double *area`, `double dz`, `int *np`)
Write gridded data in ASCII format.
- void `write_grid_nc` (`const char *filename`, `ctl_t *ctl`, `double *cd`, `double *vmr_expl`, `double *vmr_impl`, `double t`, `double *z`, `double *lon`, `double *lat`, `double *area`, `double dz`, `int *np`)
Write gridded data in netCDF format.
- int `write_met` (`char *filename`, `ctl_t *ctl`, `met_t *met`)
Read meteo data file.

- void `write_met_bin_2d` (FILE *out, `met_t` *met, float var[EX][EY], char *varname)
Write 2-D meteo variable.
- void `write_met_bin_3d` (FILE *out, `ctl_t` *ctl, `met_t` *met, float var[EX][EY][EP], char *varname, int precision, double tolerance)
Write 3-D meteo variable.
- void `write_prof` (const char *filename, `ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double t)
Write profile data.
- void `write_sample` (const char *filename, `ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double t)
Write sample data.
- void `write_station` (const char *filename, `ctl_t` *ctl, `atm_t` *atm, double t)
Write station data.

5.23.1 Detailed Description

MPTRAC library declarations.

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

5.23.2 Macro Definition Documentation

5.23.2.1 CPD `#define CPD 1003.5`

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

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

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

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

Definition at line 88 of file [libtrac.h](#).

5.23.2.3 G0 `#define G0 9.80665`

Standard gravity [m/s²].

Definition at line 93 of file [libtrac.h](#).

5.23.2.4 H0 `#define H0 7.0`

Scale height [km].

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

5.23.2.5 LV `#define LV 2501000.`

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

Definition at line 103 of file [libtrac.h](#).

5.23.2.6 KB `#define KB 1.3806504e-23`

Boltzmann constant [$\text{kg m}^2/(\text{K s}^2)$].

Definition at line 108 of file [libtrac.h](#).

5.23.2.7 MA `#define MA 28.9644`

Molar mass of dry air [g/mol].

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

5.23.2.8 MH2O `#define MH2O 18.01528`

Molar mass of water vapor [g/mol].

Definition at line 118 of file [libtrac.h](#).

5.23.2.9 MO3 `#define MO3 48.00`

Molar mass of ozone [g/mol].

Definition at line 123 of file [libtrac.h](#).

5.23.2.10 P0 `#define P0 1013.25`

Standard pressure [hPa].

Definition at line [128](#) of file [libtrac.h](#).

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

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

Definition at line [133](#) of file [libtrac.h](#).

5.23.2.12 RE `#define RE 6367.421`

Mean radius of Earth [km].

Definition at line [138](#) of file [libtrac.h](#).

5.23.2.13 RI `#define RI 8.3144598`

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

Definition at line [143](#) of file [libtrac.h](#).

5.23.2.14 T0 `#define T0 273.15`

Standard temperature [K].

Definition at line [148](#) of file [libtrac.h](#).

5.23.2.15 LEN `#define LEN 5000`

Maximum length of ASCII data lines.

Definition at line [157](#) of file [libtrac.h](#).

5.23.2.16 NP `#define NP 10000000`

Maximum number of atmospheric data points.

Definition at line 162 of file [libtrac.h](#).

5.23.2.17 NQ `#define NQ 15`

Maximum number of quantities per data point.

Definition at line 167 of file [libtrac.h](#).

5.23.2.18 NCSI `#define NCSI 1000000`

Maximum number of data points for CSI calculation.

Definition at line 172 of file [libtrac.h](#).

5.23.2.19 EP `#define EP 140`

Maximum number of pressure levels for meteo data.

Definition at line 177 of file [libtrac.h](#).

5.23.2.20 EX `#define EX 1201`

Maximum number of longitudes for meteo data.

Definition at line 182 of file [libtrac.h](#).

5.23.2.21 EY `#define EY 601`

Maximum number of latitudes for meteo data.

Definition at line 187 of file [libtrac.h](#).

5.23.2.22 NENS `#define NENS 2000`

Maximum number of data points for ensemble analysis.

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

5.23.2.23 NOBS `#define NOBS 10000000`

Maximum number of observation data points.

Definition at line [197](#) of file [libtrac.h](#).

5.23.2.24 NTHREADS `#define NTHREADS 512`

Maximum number of OpenMP threads.

Definition at line [202](#) of file [libtrac.h](#).

5.23.2.25 CY `#define CY 250`

Maximum number of latitudes for climatological data.

Definition at line [207](#) of file [libtrac.h](#).

5.23.2.26 CP `#define CP 60`

Maximum number of pressure levels for climatological data.

Definition at line [212](#) of file [libtrac.h](#).

5.23.2.27 CT `#define CT 12`

Maximum number of time steps for climatological data.

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

5.23.2.28 ALLOC `#define ALLOC(
 ptr,
 type,
 n)`

Value:

```
if((ptr=calloc((size_t)(n), sizeof(type))==NULL) \
    ERRMSG("Out of memory!");
```

Allocate and clear memory.

Definition at line 232 of file [libtrac.h](#).

5.23.2.29 ARRAY_2D `#define ARRAY_2D(
 ix,
 iy,
 ny) ((ix) * (ny) + (iy))`

Get 2-D array index.

Definition at line 238 of file [libtrac.h](#).

5.23.2.30 ARRAY_3D `#define ARRAY_3D(
 ix,
 iy,
 ny,
 iz,
 nz) (((ix)*(ny) + (iy)) * (nz) + (iz))`

Get 3-D array index.

Definition at line 242 of file [libtrac.h](#).

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

Convert degrees to zonal distance.

Definition at line 246 of file [libtrac.h](#).

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

Convert degrees to meridional distance.

Definition at line 250 of file [libtrac.h](#).

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

Convert pressure change to vertical distance.

Definition at line 254 of file [libtrac.h](#).

5.23.2.34 DX2DEG `#define DX2DEG(
 dx,
 lat)`

Value:

```
((lat) < -89.999 || (lat) > 89.999) ? 0  
: (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI))
```

 \

Convert zonal distance to degrees.

Definition at line 258 of file [libtrac.h](#).

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

Convert meridional distance to degrees.

Definition at line 263 of file [libtrac.h](#).

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

Convert vertical distance to pressure change.

Definition at line 267 of file [libtrac.h](#).

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

Compute Cartesian distance between two vectors.

Definition at line 271 of file [libtrac.h](#).

5.23.2.38 DIST2 `#define DIST2(
 a,
 b) ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))`

Compute squared distance between two vectors.

Definition at line 275 of file [libtrac.h](#).

5.23.2.39 DOTP `#define DOTP(
 a,
 b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])`

Compute dot product of two vectors.

Definition at line 279 of file [libtrac.h](#).

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

Compute floating point modulo.

Definition at line 283 of file [libtrac.h](#).

5.23.2.41 FREAD `#define FREAD(
 ptr,
 type,
 size,
 out)`

Value:

```
{
    if(fread(ptr, sizeof(type), size, out)!=size)
        ERRMSG("Error while reading!");
}
```

Read binary data.

Definition at line 287 of file [libtrac.h](#).

5.23.2.42 FWRITE `#define FWRITE(
 ptr,
 type,
 size,
 out)`

Value:

```
{
    if(fwrite(ptr, sizeof(type), size, out)!=size)
        ERRMSG("Error while writing!");
}
```

Write binary data.

Definition at line 293 of file [libtrac.h](#).

5.23.2.43 INTPOL_INIT `#define INTPOL_INIT double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};`

Initialize cache variables for interpolation.

Definition at line 299 of file [libtrac.h](#).

5.23.2.44 INTPOL_2D `#define INTPOL_2D(
 var,
 init)`

Value:

```
intpol_met_time_2d(met0, met0->var, met1, met1->var,  

    atm->time[ip], atm->lon[ip], atm->lat[ip],  

    &var, ci, cw, init);
```

2-D interpolation of a meteo variable.

Definition at line 303 of file [libtrac.h](#).

5.23.2.45 INTPOL_3D `#define INTPOL_3D(
 var,
 init)`

Value:

```
intpol_met_time_3d(met0, met0->var, met1, met1->var,  

    atm->time[ip], atm->p[ip],  

    atm->lon[ip], atm->lat[ip],  

    &var, ci, cw, init);
```

3-D interpolation of a meteo variable.

Definition at line 309 of file [libtrac.h](#).

5.23.2.46 INTPOL_SPACE_ALL #define INTPOL_SPACE_ALL(

p,
lon,
lat)

Value:

```
{
    intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1); \
    intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0); \
    intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0); \
    intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0); \
    intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0); \
    intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0); \
    intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0); \
    intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0); \
    intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0); \
    intpol_met_space_3d(met, met->iwc, p, lon, lat, &iwc, ci, cw, 0); \
    intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0); \
    intpol_met_space_2d(met, met->ts, lon, lat, &ts, ci, cw, 0); \
    intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0); \
    intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0); \
    intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0); \
    intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0); \
    intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0); \
    intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0); \
    intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0); \
    intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0); \
    intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0); \
    intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0); \
    intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0); \
    intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0); \
    intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0); \
    intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0); \
    intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0); \
    intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0); \
}
```

Spatial interpolation of all meteo data.

Definition at line 316 of file [libtrac.h](#).

5.23.2.47 INTPOL_TIME_ALL #define INTPOL_TIME_ALL(

time,
p,
lon,
lat)

Value:

```
{
    intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
    intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
    intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->cl, met1, met1->cl, time, lon, lat, &cl, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
    intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
}
```

```

intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0); \
}

```

Temporal interpolation of all meteo data.

Definition at line 348 of file [libtrac.h](#).

5.23.2.48 LAPSE `#define LAPSE(`
`p1,`
`t1,`
`p2,`
`t2)`

Value:

```

(1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1))
* ((p2) + (p1)) / ((p2) - (p1))) \

```

Calculate lapse rate between pressure levels.

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

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

Compute linear interpolation.

Definition at line 385 of file [libtrac.h](#).

5.23.2.50 NC `#define NC(`
`cmd)`

Value:

```

{ \
int nc_result=(cmd); \
if(nc_result!=NC_NOERR) \
ERRMSG("%s", nc_strerror(nc_result)); \
}

```

Execute netCDF library command and check result.

Definition at line 389 of file [libtrac.h](#).

5.23.2.51 NC_DEF_VAR `#define NC_DEF_VAR(`
`varname,`
`type,`
`ndims,`
`dims,`
`long_name,`
`units)`

Value:

```
{ \
  NC(nc_def_var(ncid, varname, type, ndims, dims, &varid)); \
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name)); \
  NC(nc_put_att_text(ncid, varid, "units", strlen(units), units)); \
}
```

Define netCDF variable.

Definition at line 396 of file [libtrac.h](#).

5.23.2.52 NC_GET_DOUBLE `#define NC_GET_DOUBLE(`
`varname,`
`ptr,`
`force)`

Value:

```
{ \
  if(force) { \
    NC(nc_inq_varid(ncid, varname, &varid)); \
    NC(nc_get_var_double(ncid, varid, ptr)); \
  } else { \
    if(nc_inq_varid(ncid, varname, &varid) == NC_NOERR) { \
      NC(nc_get_var_double(ncid, varid, ptr)); \
    } else \
      WARN("netCDF variable %s is missing!", varname); \
  } \
}
```

Read netCDF double array.

Definition at line 403 of file [libtrac.h](#).

5.23.2.53 NC_INQ_DIM `#define NC_INQ_DIM(`
`dimname,`
`ptr,`
`min,`
`max)`

Value:

```
{ \
  int dimid; size_t naux; \
  NC(nc_inq_dimid(ncid, dimname, &dimid)); \
  NC(nc_inq_dimlen(ncid, dimid, &naux)); \
  *ptr = (int)naux; \
  if ((*ptr) < (min) || (*ptr) > (max)) \
    ERRMSG("Dimension %s is out of range!", dimname); \
}
```

Read netCDF dimension.

Definition at line 416 of file [libtrac.h](#).

5.23.2.54 NC_PUT_DOUBLE #define NC_PUT_DOUBLE(
 varname,
 ptr,
 hyperslab)

Value:

```
{
    \
    NC(nc_inq_varid(ncid, varname, &varid));
    if(hyperslab) {
        NC(nc_put_vara_double(ncid, varid, start, count, ptr));
    } else {
        NC(nc_put_var_double(ncid, varid, ptr));
    }
}
```

Write netCDF double array.

Definition at line 426 of file [libtrac.h](#).

5.23.2.55 NC_PUT_INT #define NC_PUT_INT(
 varname,
 ptr,
 hyperslab)

Value:

```
{
    \
    NC(nc_inq_varid(ncid, varname, &varid));
    if(hyperslab) {
        NC(nc_put_vara_int(ncid, varid, start, count, ptr));
    } else {
        NC(nc_put_var_int(ncid, varid, ptr));
    }
}
```

Write netCDF integer array.

Definition at line 436 of file [libtrac.h](#).

5.23.2.56 NC_PUT_ATT #define NC_PUT_ATT(
 varname,
 attname,
 text)

Value:

```
{
    \
    NC(nc_inq_varid(ncid, varname, &varid));
    NC(nc_put_att_text(ncid, varid, attname, strlen(text), text));
}
```

Set netCDF attribute.

Definition at line 446 of file [libtrac.h](#).

5.23.2.57 NC_PUT_ATT_GLOBAL `#define NC_PUT_ATT_GLOBAL(
 attname,
 text) NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));`

Set netCDF global attribute.

Definition at line 452 of file [libtrac.h](#).

5.23.2.58 NC_PUT_FLOAT `#define NC_PUT_FLOAT(
 varname,
 ptr,
 hyperslab)`

Value:

```
{
    NC(nc_inq_varid(ncid, varname, &varid));
    if(hyperslab) {
        NC(nc_put_vara_float(ncid, varid, start, count, ptr));
    } else {
        NC(nc_put_var_float(ncid, varid, ptr));
    }
}
```

Write netCDF float array.

Definition at line 456 of file [libtrac.h](#).

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

Compute nearest neighbor interpolation.

Definition at line 466 of file [libtrac.h](#).

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

Compute norm of a vector.

Definition at line 470 of file [libtrac.h](#).

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

Convert altitude to pressure.

Definition at line 474 of file [libtrac.h](#).

5.23.2.62 PSAT `#define PSAT(
 t) (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))`

Compute saturation pressure over water (WMO, 2018).

Definition at line 478 of file [libtrac.h](#).

5.23.2.63 PSICE `#define PSICE(
 t) (6.112 * exp(22.46 * ((t) - T0) / (272.62 + (t) - T0)))`

Compute saturation pressure over ice (WMO, 2018).

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

5.23.2.64 PW `#define PW(
 p,
 h2o)`

Value:

```
((p) * GSL_MAX((h2o), 0.1e-6) / (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6))) \
```

Calculate partial water vapor pressure.

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

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

Compute relative humidity over water.

Definition at line 491 of file [libtrac.h](#).

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

Compute relative humidity over ice.

Definition at line 495 of file [libtrac.h](#).

5.23.2.67 RHO `#define RHO(`
`p,`
`t) (100. * (p) / (RA * (t)))`

Compute density of air.

Definition at line 499 of file [libtrac.h](#).

5.23.2.68 SET_ATM `#define SET_ATM(`
`qnt,`
`val)`

Value:

```
if (ctl->qnt >= 0) \
    atm->q[ctl->qnt][ip] = val;
```

Set atmospheric quantity value.

Definition at line 503 of file [libtrac.h](#).

5.23.2.69 SET_QNT `#define SET_QNT(`
`qnt,`
`name,`
`longname,`
`unit)`

Value:

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

Set atmospheric quantity index.

Definition at line 508 of file [libtrac.h](#).

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

Compute specific humidity from water vapor volume mixing ratio.

Definition at line 516 of file [libtrac.h](#).

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

Compute square.

Definition at line 520 of file [libtrac.h](#).

5.23.2.72 SWAP `#define SWAP(
 x,
 y,
 type) do {type tmp = x; x = y; y = tmp;} while(0);`

Swap macro.

Definition at line 524 of file [libtrac.h](#).

5.23.2.73 TDEW `#define TDEW(
 p,
 h2o)`

Value:

`(T0 + 243.12 * log(PW((p), (h2o)) / 6.112)
 / (17.62 - log(PW((p), (h2o)) / 6.112))) \`

Calculate dew point temperature (WMO, 2018).

Definition at line 528 of file [libtrac.h](#).

5.23.2.74 TICE `#define TICE(
 p,
 h2o)`

Value:

`(T0 + 272.62 * log(PW((p), (h2o)) / 6.112)
 / (22.46 - log(PW((p), (h2o)) / 6.112))) \`

Calculate frost point temperature (WMO, 2018).

Definition at line 533 of file [libtrac.h](#).

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

Compute potential temperature.

Definition at line 538 of file [libtrac.h](#).

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

Compute virtual potential temperature.

Definition at line 542 of file [libtrac.h](#).

5.23.2.77 TOK `#define TOK(
 line,
 tok,
 format,
 var)`

Value:

```
{
    if(((tok)=strtok((line), " \t"))) {
        if(sscanf(tok, format, &(var))!=1) continue;
    } else ERRMSG("Error while reading!");
}
```

Get string tokens.

Definition at line 546 of file [libtrac.h](#).

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

Compute virtual temperature.

Definition at line 553 of file [libtrac.h](#).

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

Convert pressure to altitude.

Definition at line 557 of file [libtrac.h](#).

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

Value:

```
(RI / MA / GO * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1))) \
 * ((lnp0) - (lnp1)))
```

Calculate geopotential height difference.

Definition at line 561 of file [libtrac.h](#).

5.23.2.81 ZETA `#define ZETA(
 ps,
 p,
 t)`

Value:

```
((p) / (ps) <= 0.3 ? 1. : \
 sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3))) \
 * THETA((p), (t)))
```

Calculate zeta vertical coordinate.

Definition at line 566 of file [libtrac.h](#).

5.23.2.82 LOGLEV `#define LOGLEV 2`

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

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

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

Value:

```
{ \
 if(level >= 2) \
 printf(" "); \
 if(level <= LOGLEV) { \
 printf(__VA_ARGS__); \
 printf("\n"); \
 } \
}
```

Print log message.

Definition at line 581 of file [libtrac.h](#).

5.23.2.84 WARN `#define WARN(
...)`

Value:

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

Print warning message.

Definition at line 591 of file [libtrac.h](#).

5.23.2.85 ERRMSG `#define ERRMSG(
...)`

Value:

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

Print error message and quit program.

Definition at line 597 of file [libtrac.h](#).

5.23.2.86 PRINT `#define PRINT(
 format,
 var)`

Value:

```
printf("Print (%s, %s, l%d): %s= \"format\"\n", \
    __FILE__, __func__, __LINE__, #var, var);
```

Print macro for debugging.

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

5.23.2.87 NTIMER `#define NTIMER 100`

Maximum number of timers.

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

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

Print timers.

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

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

Value:

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

Select timer.

Definition at line [620](#) of file [libtrac.h](#).

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

Start timers.

Definition at line [627](#) of file [libtrac.h](#).

5.23.2.91 STOP_TIMERS `#define STOP_TIMERS NVTX_POP;`

Stop timers.

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

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

Definition at line [678](#) of file [libtrac.h](#).

5.23.2.93 NVTX_POP `#define NVTX_POP {}`

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

5.23.3 Function Documentation

5.23.3.1 thrustSortWrapper() void thrustSortWrapper (

```
double *__restrict__ c,
int n,
int *__restrict__ index )
```

Wrapper to Thrust sorting function.

5.23.3.2 buoyancy_frequency() double buoyancy_frequency (

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

Calculate buoyancy frequency.

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

```
00033     {
00034
00035     double theta0 = THETA(p0, t0);
00036     double theta1 = THETA(p1, t1);
00037     double dz = RI / MA / GO * 0.5 * (t0 + t1) * (log(p0) - log(p1));
00038
00039     return sqrt(2. * GO / (theta0 + theta1) * (theta1 - theta0) / dz);
00040 }
```

5.23.3.3 cart2geo() void cart2geo (

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

Convert Cartesian coordinates to geolocation.

Definition at line 44 of file [libtrac.c](#).

```
00048     {
00049
00050     double radius = NORM(x);
00051     *lat = asin(x[2] / radius) * 180. / M_PI;
00052     *lon = atan2(x[1], x[0]) * 180. / M_PI;
00053     *z = radius - RE;
00054 }
```

5.23.3.4 check_finite() int check_finite (

```
const double x )
```

Check if x is finite.

```

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

```

Climatology of HNO3 volume mixing ratios.

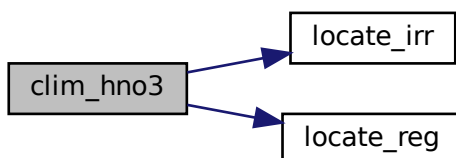
Definition at line 58 of file `libtrac.c`.

```

00062     {
00063
00064         /* Get seconds since begin of year... */
00065         double sec = FMOD(t, 365.25 * 86400.);
00066         while (sec < 0)
00067             sec += 365.25 * 86400.;
00068
00069         /* Check pressure... */
00070         if (p < clim->hno3_p[0])
00071             p = clim->hno3_p[0];
00072         else if (p > clim->hno3_p[clim->hno3_np - 1])
00073             p = clim->hno3_p[clim->hno3_np - 1];
00074
00075         /* Check latitude... */
00076         if (lat < clim->hno3_lat[0])
00077             lat = clim->hno3_lat[0];
00078         else if (lat > clim->hno3_lat[clim->hno3_nlat - 1])
00079             lat = clim->hno3_lat[clim->hno3_nlat - 1];
00080
00081         /* Get indices... */
00082         int isec = locate_irr(clim->hno3_time, clim->hno3_ntime, sec);
00083         int ilat = locate_reg(clim->hno3_lat, clim->hno3_nlat, lat);
00084         int ip = locate_irr(clim->hno3_p, clim->hno3_np, p);
00085
00086         /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00087         double aux00 = LIN(clim->hno3_p[ip],
00088                             clim->hno3[isec][ilat][ip],
00089                             clim->hno3_p[ip + 1],
00090                             clim->hno3[isec][ilat][ip + 1], p);
00091         double aux01 = LIN(clim->hno3_p[ip],
00092                             clim->hno3[isec][ilat + 1][ip],
00093                             clim->hno3_p[ip + 1],
00094                             clim->hno3[isec][ilat + 1][ip + 1], p);
00095         double aux10 = LIN(clim->hno3_p[ip],
00096                             clim->hno3[isec + 1][ilat][ip],
00097                             clim->hno3_p[ip + 1],
00098                             clim->hno3[isec + 1][ilat][ip + 1], p);
00099         double aux11 = LIN(clim->hno3_p[ip],
00100                             clim->hno3[isec + 1][ilat + 1][ip],
00101                             clim->hno3_p[ip + 1],
00102                             clim->hno3[isec + 1][ilat + 1][ip + 1], p);
00103         aux00 = LIN(clim->hno3_lat[ilat], aux00,
00104                     clim->hno3_lat[ilat + 1], aux01, lat);
00105         aux11 = LIN(clim->hno3_lat[ilat], aux10,
00106                     clim->hno3_lat[ilat + 1], aux11, lat);
00107         aux00 = LIN(clim->hno3_time[isec], aux00,
00108                     clim->hno3_time[isec + 1], aux11, sec);
00109
00110         /* Convert from ppb to ppv... */
00111         return GSL_MAX(1e-9 * aux00, 0.0);
00112     }

```

Here is the call graph for this function:



5.23.3.6 clim_hno3_init() void clim_hno3_init (clim_t * clim)

Initialization function for HNO3 climatology.

Definition at line 116 of file libtrac.c.

```

00117     {
00118
00119     /* Write info... */
00120     LOG(1, "Initialize HNO3 data...");
00121
00122     clim->hno3_nptime = 12;
00123     double hno3_time[12] = {
00124         1209600.00, 3888000.00, 6393600.00,
00125         9072000.00, 11664000.00, 14342400.00,
00126         16934400.00, 19612800.00, 22291200.00,
00127         24883200.00, 27561600.00, 30153600.00
00128     };
00129     memcpy(clim->hno3_time, hno3_time, sizeof(clim->hno3_time));
00130
00131     clim->hno3_nlat = 18;
00132     double hno3_lat[18] = {
00133         -85, -75, -65, -55, -45, -35, -25, -15, -5,
00134         5, 15, 25, 35, 45, 55, 65, 75, 85
00135     };
00136     memcpy(clim->hno3_lat, hno3_lat, sizeof(clim->hno3_lat));
00137
00138     clim->hno3_np = 10;
00139     double hno3_p[10] = {
00140         4.64159, 6.81292, 10, 14.678, 21.5443,
00141         31.6228, 46.4159, 68.1292, 100, 146.78
00142     };
00143     memcpy(clim->hno3_p, hno3_p, sizeof(clim->hno3_p));
00144
00145     double hno3[12][18][10] = {
00146         {0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00147         {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00148         {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00149         {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00150         {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00151         {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00152         {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00153         {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00154         {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00155         {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00156         {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00157         {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00158         {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00159         {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00160         {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00161         {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00162         {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00163         {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19},
00164         {0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00165         {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00166         {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00167         {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00168         {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00169         {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00170         {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00171         {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00172         {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00173         {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00174         {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00175         {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00176         {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00177         {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00178         {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00179         {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00180         {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00181         {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17},
00182         {1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00183         {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00184         {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00185         {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00186         {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00187         {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00188         {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00189         {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00190         {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00191         {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00192         {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00193         {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00194         {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00195         {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},

```

```
00196 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00197 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00198 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00199 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00200 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75}},
00201 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58}},
00202 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5}},
00203 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09}},
00204 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727}},
00205 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409}},
00206 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198}},
00207 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12}},
00208 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172}},
00209 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157}},
00210 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138}},
00211 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286}},
00212 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02}},
00213 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96}},
00214 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52}},
00215 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04}},
00216 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46}},
00217 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00218 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63}},
00219 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57}},
00220 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63}},
00221 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37}},
00222 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88}},
00223 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527}},
00224 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229}},
00225 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972}},
00226 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126}},
00227 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183}},
00228 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18}},
00229 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343}},
00230 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964}},
00231 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83}},
00232 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25}},
00233 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39}},
00234 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52}},
00235 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
00236 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26}},
00237 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05}},
00238 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65}},
00239 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67}},
00240 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13}},
00241 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639}},
00242 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217}},
00243 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694}},
00244 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136}},
00245 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194}},
00246 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229}},
00247 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302}},
00248 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66}},
00249 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41}},
00250 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8}},
00251 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9}},
00252 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88}},
00253 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
00254 {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33}},
00255 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78}},
00256 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08}},
00257 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3}},
00258 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38}},
00259 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656}},
00260 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176}},
00261 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705}},
00262 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12}},
00263 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199}},
00264 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25}},
00265 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259}},
00266 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422}},
00267 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913}},
00268 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4}},
00269 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56}},
00270 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61}},
00271 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
00272 {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4}},
00273 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73}},
00274 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6}},
00275 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14}},
00276 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38}},
00277 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672}},
00278 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19}},
00279 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181}},
00280 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107}},
00281 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185}},
00282 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224}},
```

```

00283     {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
00284     {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
00285     {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
00286     {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
00287     {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
00288     {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
00289     {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
00290     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
00291     {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74}},
00292     {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
00293     {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
00294     {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
00295     {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
00296     {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
00297     {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
00298     {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
00299     {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
00300     {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
00301     {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
00302     {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
00303     {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
00304     {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
00305     {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
00306     {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
00307     {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
00308     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
00309     {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84}},
00310     {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
00311     {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
00312     {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
00313     {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
00314     {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
00315     {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
00316     {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
00317     {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
00318     {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
00319     {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
00320     {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
00321     {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
00322     {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
00323     {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
00324     {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
00325     {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
00326     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
00327     {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
00328     {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
00329     {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
00330     {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
00331     {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
00332     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00333     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00334     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00335     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00336     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00337     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00338     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00339     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00340     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00341     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00342     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00343     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
00344     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00345     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00346     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00347     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00348     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00349     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00350     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00351     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00352     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00353     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00354     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00355     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00356     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00357     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00358     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00359     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00360     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00361     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00362 };
00363 memcpy(clim->hno3, hno3, sizeof(clim->hno3));
00364
00365 /* Get range... */
00366 double hno3min = 1e99, hno3max = -1e99;
00367 for (int it = 0; it < clim->hno3_ntime; it++)
00368     for (int iz = 0; iz < clim->hno3_np; iz++)
00369         for (int iy = 0; iy < clim->hno3_nlat; iy++) {

```

```

00370         hno3min = GSL_MIN(hno3min, clim->hno3[it][iy][iz]);
00371         hno3max = GSL_MAX(hno3max, clim->hno3[it][iy][iz]);
00372     }
00373
00374     /* Write info... */
00375     LOG(2, "Number of time steps: %d", clim->hno3_ntime);
00376     LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00377         clim->hno3_time[0], clim->hno3_time[1],
00378         clim->hno3_time[clim->hno3_ntime - 1]);
00379     LOG(2, "Number of pressure levels: %d", clim->hno3_np);
00380     LOG(2, "Altitude levels: %g, %g ... %g km",
00381         Z(clim->hno3_p[0]), Z(clim->hno3_p[1]),
00382         Z(clim->hno3_p[clim->hno3_np - 1]));
00383     LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->hno3_p[0],
00384         clim->hno3_p[1], clim->hno3_p[clim->hno3_np - 1]);
00385     LOG(2, "Number of latitudes: %d", clim->hno3_nlat);
00386     LOG(2, "Latitudes: %g, %g ... %g deg",
00387         clim->hno3_lat[0], clim->hno3_lat[1],
00388         clim->hno3_lat[clim->hno3_nlat - 1]);
00389     LOG(2, "HNO3 concentration range: %g ... %g ppv", 1e-9 * hno3min,
00390         1e-9 * hno3max);
00391 }

```

5.23.3.7 clim_oh() double clim_oh (

```

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

```

Climatology of OH number concentrations.

Definition at line 395 of file libtrac.c.

```

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

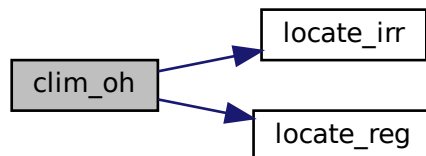
```

```

00444
00445     return GSL_MAX(aux00, 0.0);
00446 }

```

Here is the call graph for this function:



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

Climatology of OH number concentrations with diurnal variation.

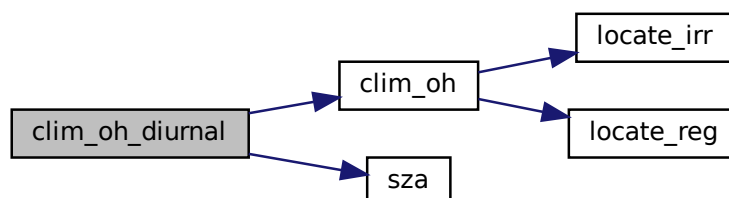
Definition at line 450 of file [libtrac.c](#).

```

00456     {
00457
00458     double oh = clim_oh(clim, t, lat, p), sza2 = sza(t, lon, lat);
00459
00460     if (sza2 <= M_PI / 2. * 89. / 90.)
00461         return oh * exp(-ctl->oh_chem_beta / cos(sza2));
00462     else
00463         return oh * exp(-ctl->oh_chem_beta / cos(M_PI / 2. * 89. / 90.));
00464 }

```

Here is the call graph for this function:



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

Initialization function for OH climatology.

Definition at line 468 of file libtrac.c.

```
00470     {
00471
00472     int nt, ncid, varid;
00473
00474     double *help, ohmin = 1e99, ohmax = -1e99;
00475
00476     /* Write info... */
00477     LOG(1, "Read OH data: %s", ctl->clim_oh_filename);
00478
00479     /* Open netCDF file... */
00480     if (nc_open(ctl->clim_oh_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00481         WARN("OH climatology data are missing!");
00482         return;
00483     }
00484
00485     /* Read pressure data... */
00486     NC_INQ_DIM("press", &clim->oh_np, 2, CP);
00487     NC_GET_DOUBLE("press", clim->oh_p, 1);
00488
00489     /* Check ordering of pressure data... */
00490     if (clim->oh_p[0] < clim->oh_p[1])
00491         ERRMSG("Pressure data are not descending!");
00492
00493     /* Read latitudes... */
00494     NC_INQ_DIM("lat", &clim->oh_nlat, 2, CY);
00495     NC_GET_DOUBLE("lat", clim->oh_lat, 1);
00496
00497     /* Check ordering of latitudes... */
00498     if (clim->oh_lat[0] > clim->oh_lat[1])
00499         ERRMSG("Latitude data are not ascending!");
00500
00501     /* Set time data for monthly means... */
00502     clim->oh_ntime = 12;
00503     clim->oh_time[0] = 1209600.00;
00504     clim->oh_time[1] = 3888000.00;
00505     clim->oh_time[2] = 6393600.00;
00506     clim->oh_time[3] = 9072000.00;
00507     clim->oh_time[4] = 11664000.00;
00508     clim->oh_time[5] = 14342400.00;
00509     clim->oh_time[6] = 16934400.00;
00510     clim->oh_time[7] = 19612800.00;
00511     clim->oh_time[8] = 22291200.00;
00512     clim->oh_time[9] = 24883200.00;
00513     clim->oh_time[10] = 27561600.00;
00514     clim->oh_time[11] = 30153600.00;
00515
00516     /* Check number of timesteps... */
00517     NC_INQ_DIM("time", &nt, 12, 12);
00518
00519     /* Read OH data... */
00520     ALLOC(help, double,
00521           clim->oh_nlat * clim->oh_np * clim->oh_ntime);
00522     NC_GET_DOUBLE("OH", help, 1);
00523     for (int it = 0; it < clim->oh_ntime; it++)
00524         for (int iz = 0; iz < clim->oh_np; iz++)
00525             for (int iy = 0; iy < clim->oh_nlat; iy++) {
00526                 clim->oh[it][iz][iy] =
00527                     help[ARRAY_3D(it, iz, clim->oh_np, iy, clim->oh_nlat)]
00528                     / clim_oh_init_help(ctl->oh_chem_beta, clim->oh_time[it],
00529                                         clim->oh_lat[iy]);
00530                 ohmin = GSL_MIN(ohmin, clim->oh[it][iz][iy]);
00531                 ohmax = GSL_MAX(ohmax, clim->oh[it][iz][iy]);
00532             }
00533     free(help);
00534
00535     /* Close netCDF file... */
00536     NC(nc_close(ncid));
00537
00538     /* Write info... */
00539     LOG(2, "Number of time steps: %d", clim->oh_ntime);
00540     LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00541         clim->oh_time[0], clim->oh_time[1], clim->oh_time[clim->oh_ntime - 1]);
00542     LOG(2, "Number of pressure levels: %d", clim->oh_np);
00543     LOG(2, "Altitude levels: %g, %g ... %g km",
00544         Z(clim->oh_p[0]), Z(clim->oh_p[1]), Z(clim->oh_p[clim->oh_np - 1]));
00545     LOG(2, "Pressure levels: %g, %g ... %g hPa",
00546         clim->oh_p[0], clim->oh_p[1], clim->oh_p[clim->oh_np - 1]);
```

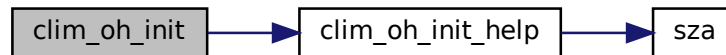


```

00547 LOG(2, "Number of latitudes: %d", clim->oh_nlat);
00548 LOG(2, "Latitudes: %g, %g ... %g deg",
00549      clim->oh_lat[0], clim->oh_lat[1], clim->oh_lat[clim->oh_nlat - 1]);
00550 LOG(2, "OH concentration range: %g ... %g molec/cm^3", ohmin, ohmax);
00551 }

```

Here is the call graph for this function:



5.23.3.10 clim_oh_init_help() double clim_oh_init_help (

```

double beta,
double time,
double lat )

```

Apply diurnal correction to OH climatology.

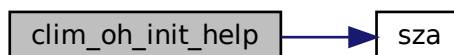
Definition at line 555 of file libtrac.c.

```

00558 {
00559
00560     double aux, lon, sum = 0;
00561
00562     int n = 0;
00563
00564     /* Integrate day/night correction factor over longitude... */
00565     for (lon = -180; lon < 180; lon += 1) {
00566         aux = sza(time, lon, lat);
00567         if (aux <= M_PI / 2. * 85. / 90.)
00568             sum += exp(-beta / cos(aux));
00569         else
00570             sum += exp(-beta / cos(M_PI / 2. * 85. / 90.));
00571         n++;
00572     }
00573     return sum / (double) n;
00574 }

```

Here is the call graph for this function:



```

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

```

Climatology of H2O2 number concentrations.

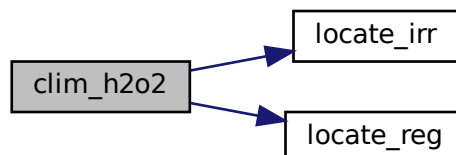
Definition at line 578 of file libtrac.c.

```

00582     {
00583
00584     /* Get seconds since begin of year... */
00585     double sec = FMOD(t, 365.25 * 86400.);
00586     while (sec < 0)
00587         sec += 365.25 * 86400.;
00588
00589     /* Check pressure... */
00590     if (p < clim->h2o2_p[clim->h2o2_np - 1])
00591         p = clim->h2o2_p[clim->h2o2_np - 1];
00592     else if (p > clim->h2o2_p[0])
00593         p = clim->h2o2_p[0];
00594
00595     /* Check latitude... */
00596     if (lat < clim->h2o2_lat[0])
00597         lat = clim->h2o2_lat[0];
00598     else if (lat > clim->h2o2_lat[clim->h2o2_nlat - 1])
00599         lat = clim->h2o2_lat[clim->h2o2_nlat - 1];
00600
00601     /* Get indices... */
00602     int isec = locate_irr(clim->h2o2_time, clim->h2o2_ntime, sec);
00603     int ilat = locate_reg(clim->h2o2_lat, clim->h2o2_nlat, lat);
00604     int ip = locate_irr(clim->h2o2_p, clim->h2o2_np, p);
00605
00606     /* Interpolate H2O2 climatology... */
00607     double aux00 = LIN(clim->h2o2_p[ip],
00608                       clim->h2o2[isec][ip][ilat],
00609                       clim->h2o2_p[ip + 1],
00610                       clim->h2o2[isec][ip + 1][ilat], p);
00611     double aux01 = LIN(clim->h2o2_p[ip],
00612                       clim->h2o2[isec][ip][ilat + 1],
00613                       clim->h2o2_p[ip + 1],
00614                       clim->h2o2[isec][ip + 1][ilat + 1], p);
00615     double aux10 = LIN(clim->h2o2_p[ip],
00616                       clim->h2o2[isec + 1][ip][ilat],
00617                       clim->h2o2_p[ip + 1],
00618                       clim->h2o2[isec + 1][ip + 1][ilat], p);
00619     double aux11 = LIN(clim->h2o2_p[ip],
00620                       clim->h2o2[isec + 1][ip][ilat + 1],
00621                       clim->h2o2_p[ip + 1],
00622                       clim->h2o2[isec + 1][ip + 1][ilat + 1], p);
00623     aux00 =
00624         LIN(clim->h2o2_lat[ilat], aux00, clim->h2o2_lat[ilat + 1], aux01, lat);
00625     aux11 =
00626         LIN(clim->h2o2_lat[ilat], aux10, clim->h2o2_lat[ilat + 1], aux11, lat);
00627     aux00 =
00628         LIN(clim->h2o2_time[isec], aux00, clim->h2o2_time[isec + 1], aux11, sec);
00629
00630     return GSL_MAX(aux00, 0.0);
00631 }

```

Here is the call graph for this function:



5.23.3.12 clim_h2o2_init() void clim_h2o2_init (
 ctl_t * ctl,
 clim_t * clim)

Initialization function for H2O2 climatology.

Definition at line 635 of file libtrac.c.

```
00637     {
00638
00639         int ncid, varid, it, iy, iz, nt;
00640
00641         double *help, h2o2min = 1e99, h2o2max = -1e99;
00642
00643         /* Write info... */
00644         LOG(1, "Read H2O2 data: %s", ctl->clim_h2o2_filename);
00645
00646         /* Open netCDF file... */
00647         if (nc_open(ctl->clim_h2o2_filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00648             WARN("H2O2 climatology data are missing!");
00649             return;
00650         }
00651
00652         /* Read pressure data... */
00653         NC_INQ_DIM("press", &clim->h2o2_np, 2, CP);
00654         NC_GET_DOUBLE("press", clim->h2o2_p, 1);
00655
00656         /* Check ordering of pressure data... */
00657         if (clim->h2o2_p[0] < clim->h2o2_p[1])
00658             ERRMSG("Pressure data are not descending!");
00659
00660         /* Read latitudes... */
00661         NC_INQ_DIM("lat", &clim->h2o2_nlat, 2, CY);
00662         NC_GET_DOUBLE("lat", clim->h2o2_lat, 1);
00663
00664         /* Check ordering of latitude data... */
00665         if (clim->h2o2_lat[0] > clim->h2o2_lat[1])
00666             ERRMSG("Latitude data are not ascending!");
00667
00668         /* Set time data (for monthly means)... */
00669         clim->h2o2_ntime = 12;
00670         clim->h2o2_time[0] = 1209600.00;
00671         clim->h2o2_time[1] = 3888000.00;
00672         clim->h2o2_time[2] = 6393600.00;
00673         clim->h2o2_time[3] = 9072000.00;
00674         clim->h2o2_time[4] = 11664000.00;
00675         clim->h2o2_time[5] = 14342400.00;
00676         clim->h2o2_time[6] = 16934400.00;
00677         clim->h2o2_time[7] = 19612800.00;
00678         clim->h2o2_time[8] = 22291200.00;
00679         clim->h2o2_time[9] = 24883200.00;
00680         clim->h2o2_time[10] = 27561600.00;
00681         clim->h2o2_time[11] = 30153600.00;
00682
00683         /* Check number of timesteps... */
00684         NC_INQ_DIM("time", &nt, 12, 12);
00685
00686         /* Read data... */
00687         ALLOC(help, double,
00688             clim->h2o2_nlat * clim->h2o2_np * clim->h2o2_ntime);
00689         NC_GET_DOUBLE("h2o2", help, 1);
00690         for (it = 0; it < clim->h2o2_ntime; it++)
00691             for (iz = 0; iz < clim->h2o2_np; iz++)
00692                 for (iy = 0; iy < clim->h2o2_nlat; iy++) {
00693                     clim->h2o2[it][iz][iy] =
00694                         help[ARRAY_3D(it, iz, clim->h2o2_np, iy, clim->h2o2_nlat)];
00695                     h2o2min = GSL_MIN(h2o2min, clim->h2o2[it][iz][iy]);
00696                     h2o2max = GSL_MAX(h2o2max, clim->h2o2[it][iz][iy]);
00697                 }
00698         free(help);
00699
00700         /* Close netCDF file... */
00701         NC(nc_close(ncid));
00702
00703         /* Write info... */
00704         LOG(2, "Number of time steps: %d", clim->h2o2_ntime);
00705         LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00706             clim->h2o2_time[0], clim->h2o2_time[1],
00707             clim->h2o2_time[clim->h2o2_ntime - 1]);
00708         LOG(2, "Number of pressure levels: %d", clim->h2o2_np);
00709         LOG(2, "Altitude levels: %g, %g ... %g km",
00710             Z(clim->h2o2_p[0]), Z(clim->h2o2_p[1]),
00711             Z(clim->h2o2_p[clim->h2o2_np - 1]));
00712         LOG(2, "Pressure levels: %g, %g ... %g hPa", clim->h2o2_p[0],
00713             clim->h2o2_p[1], clim->h2o2_p[clim->h2o2_np - 1]);
```

```

00714 LOG(2, "Number of latitudes: %d", clim->h2o2_nlat);
00715 LOG(2, "Latitudes: %g, %g ... %g deg",
00716     clim->h2o2_lat[0], clim->h2o2_lat[1],
00717     clim->h2o2_lat[clim->h2o2_nlat - 1]);
00718 LOG(2, "H2O2 concentration range: %g ... %g molec/cm^3", h2o2min, h2o2max);
00719 }

```

5.23.3.13 clim_tropo() double clim_tropo (

```

    clim_t * clim,
    double t,
    double lat )

```

Climatology of tropopause pressure.

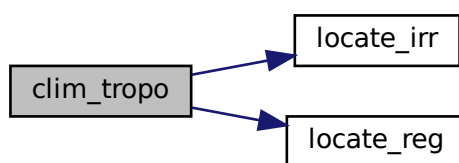
Definition at line 723 of file [libtrac.c](#).

```

00726 {
00727
00728     /* Get seconds since begin of year... */
00729     double sec = FMOD(t, 365.25 * 86400.);
00730     while (sec < 0)
00731         sec += 365.25 * 86400.;
00732
00733     /* Get indices... */
00734     int isec = locate_irr(clim->tropo_time, clim->tropo_ntime, sec);
00735     int ilat = locate_reg(clim->tropo_lat, clim->tropo_nlat, lat);
00736
00737     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
00738     double p0 = LIN(clim->tropo_lat[ilat],
00739                    clim->tropo[isec][ilat],
00740                    clim->tropo_lat[ilat + 1],
00741                    clim->tropo[isec][ilat + 1], lat);
00742     double p1 = LIN(clim->tropo_lat[ilat],
00743                    clim->tropo[isec + 1][ilat],
00744                    clim->tropo_lat[ilat + 1],
00745                    clim->tropo[isec + 1][ilat + 1], lat);
00746     return LIN(clim->tropo_time[isec], p0, clim->tropo_time[isec + 1], p1, sec);
00747 }

```

Here is the call graph for this function:



5.23.3.14 clim_tropo_init() void clim_tropo_init (

```

    clim_t * clim )

```

Initialize tropopause climatology.

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

```

00752 {

```

```

00753
00754 /* Write info... */
00755 LOG(1, "Initialize tropopause data...");
00756
00757 clim->tropo_ntime = 12;
00758 double tropo_time[12] = {
00759     1209600.00, 3888000.00, 6393600.00,
00760     9072000.00, 11664000.00, 14342400.00,
00761     16934400.00, 19612800.00, 22291200.00,
00762     24883200.00, 27561600.00, 30153600.00
00763 };
00764 memcpy(clim->tropo_time, tropo_time, sizeof(clim->tropo_time));
00765
00766 clim->tropo_nlat = 73;
00767 double tropo_lat[73] = {
00768     -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
00769     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
00770     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
00771     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
00772     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
00773     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
00774     75, 77.5, 80, 82.5, 85, 87.5, 90
00775 };
00776 memcpy(clim->tropo_lat, tropo_lat, sizeof(clim->tropo_lat));
00777
00778 double tropo[12][73] = {
00779     {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
00780     297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
00781     175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
00782     99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
00783     98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
00784     152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
00785     277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
00786     275.3, 275.6, 275.4, 274.1, 273.5},
00787     {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
00788     300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
00789     150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
00790     98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
00791     98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
00792     220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
00793     284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
00794     287.5, 286.2, 285.8},
00795     {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
00796     297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
00797     161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
00798     100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
00799     99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
00800     186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
00801     279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
00802     304.3, 304.9, 306, 306.6, 306.2, 306},
00803     {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
00804     290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
00805     195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
00806     102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
00807     99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
00808     148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
00809     263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
00810     315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
00811     {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
00812     260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
00813     205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
00814     101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
00815     102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
00816     165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
00817     273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
00818     325.3, 325.8, 325.8},
00819     {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
00820     222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
00821     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
00822     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
00823     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
00824     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
00825     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
00826     308.5, 312.2, 313.1, 313.3},
00827     {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
00828     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
00829     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
00830     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
00831     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
00832     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
00833     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
00834     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
00835     {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
00836     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
00837     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
00838     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
00839     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,

```

```

00840     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
00841     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
00842     278.2, 282.6, 287.4, 290.9, 292.5, 293},
00843     {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
00844     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
00845     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
00846     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
00847     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
00848     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
00849     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
00850     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
00851     {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
00852     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
00853     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
00854     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
00855     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
00856     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
00857     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
00858     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
00859     305.1},
00860     {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
00861     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
00862     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
00863     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
00864     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
00865     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
00866     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
00867     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
00868     {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
00869     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
00870     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
00871     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
00872     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
00873     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
00874     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
00875     281.7, 281.1, 281.2}
00876 };
00877 memcpy(clim->tropo, tropo, sizeof(clim->tropo));
00878
00879 /* Get range... */
00880 double tropomin = 1e99, tropomax = -1e99;
00881 for (int it = 0; it < clim->tropo_ntime; it++)
00882     for (int iy = 0; iy < clim->tropo_nlat; iy++) {
00883         tropomin = GSL_MIN(tropomin, clim->tropo[it][iy]);
00884         tropomax = GSL_MAX(tropomax, clim->tropo[it][iy]);
00885     }
00886
00887 /* Write info... */
00888 LOG(2, "Number of time steps: %d", clim->tropo_ntime);
00889 LOG(2, "Time steps: %.2f, %.2f ... %.2f s",
00890     clim->tropo_time[0], clim->tropo_time[1],
00891     clim->tropo_time[clim->tropo_ntime - 1]);
00892 LOG(2, "Number of latitudes: %d", clim->tropo_nlat);
00893 LOG(2, "Latitudes: %g, %g ... %g deg",
00894     clim->tropo_lat[0], clim->tropo_lat[1],
00895     clim->tropo_lat[clim->tropo_nlat - 1]);
00896 LOG(2, "Tropopause altitude range: %g ... %g hPa", Z(tropomax),
00897     Z(tropomin));
00898 LOG(2, "Tropopause pressure range: %g ... %g hPa", tropomin, tropomax);
00899 }

```

5.23.3.15 compress_pack() void compress_pack (

```

    char * varname,
    float * array,
    size_t nxy,
    size_t nz,
    int decompress,
    FILE * inout )

```

Pack or unpack array.

Definition at line 903 of file libtrac.c.

```

00909     {
00910
00911         double min[EP], max[EP], off[EP], scl[EP];
00912

```

```

00913 unsigned short *sarray;
00914
00915 /* Allocate... */
00916 ALLOC(sarray, unsigned short,
00917       nxy * nz);
00918
00919 /* Read compressed stream and decompress array... */
00920 if (decompress) {
00921
00922     /* Write info... */
00923     LOG(2, "Read 3-D variable: %s (pack, RATIO= %g %%)",
00924         varname, 100. * sizeof(unsigned short) / sizeof(float));
00925
00926     /* Read data... */
00927     FREAD(&scl, double,
00928          nz,
00929          inout);
00930     FREAD(&off, double,
00931          nz,
00932          inout);
00933     FREAD(sarray, unsigned short,
00934          nxy * nz,
00935          inout);
00936
00937     /* Convert to float... */
00938     #pragma omp parallel for default(shared)
00939     for (size_t ixy = 0; ixy < nxy; ixy++)
00940         for (size_t iz = 0; iz < nz; iz++)
00941             array[ixy * nz + iz]
00942                 = (float) (sarray[ixy * nz + iz] * scl[iz] + off[iz]);
00943 }
00944
00945 /* Compress array and output compressed stream... */
00946 else {
00947
00948     /* Write info... */
00949     LOG(2, "Write 3-D variable: %s (pack, RATIO= %g %%)",
00950         varname, 100. * sizeof(unsigned short) / sizeof(float));
00951
00952     /* Get range... */
00953     for (size_t iz = 0; iz < nz; iz++) {
00954         min[iz] = array[iz];
00955         max[iz] = array[iz];
00956     }
00957     for (size_t ixy = 1; ixy < nxy; ixy++)
00958         for (size_t iz = 0; iz < nz; iz++) {
00959             if (array[ixy * nz + iz] < min[iz])
00960                 min[iz] = array[ixy * nz + iz];
00961             if (array[ixy * nz + iz] > max[iz])
00962                 max[iz] = array[ixy * nz + iz];
00963         }
00964
00965     /* Get offset and scaling factor... */
00966     for (size_t iz = 0; iz < nz; iz++) {
00967         scl[iz] = (max[iz] - min[iz]) / 65533.;
00968         off[iz] = min[iz];
00969     }
00970
00971     /* Convert to short... */
00972     #pragma omp parallel for default(shared)
00973     for (size_t ixy = 0; ixy < nxy; ixy++)
00974         for (size_t iz = 0; iz < nz; iz++)
00975             if (scl[iz] != 0)
00976                 sarray[ixy * nz + iz] = (unsigned short)
00977                     ((array[ixy * nz + iz] - off[iz]) / scl[iz] + .5);
00978             else
00979                 sarray[ixy * nz + iz] = 0;
00980
00981     /* Write data... */
00982     FWRITE(&scl, double,
00983          nz,
00984          inout);
00985     FWRITE(&off, double,
00986          nz,
00987          inout);
00988     FWRITE(sarray, unsigned short,
00989          nxy * nz,
00990          inout);
00991 }
00992
00993 /* Free... */
00994 free(sarray);
00995 }

```

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

Compress or decompress array with zfp.

Compress or decompress array with zstd.

Get day of year from date.

Definition at line 1144 of file libtrac.c.

```
01148     {  
01149  
01150     const int  
01151     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },  
01152     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };  
01153  
01154     /* Get day of year... */  
01155     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))  
01156         *doy = d0l[mon - 1] + day - 1;  
01157     else  
01158         *doy = d0[mon - 1] + day - 1;  
01159 }
```

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

Get date from day of year.

Definition at line 1163 of file libtrac.c.

```
01167     {  
01168  
01169     const int  
01170     d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 },  
01171     d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };  
01172  
01173     int i;  
01174  
01175     /* Get month and day... */  
01176     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {  
01177         for (i = 11; i > 0; i--)  
01178             if (d0l[i] <= doy)  
01179                 break;  
01180         *mon = i + 1;  
01181         *day = doy - d0l[i] + 1;  
01182     } else {  
01183         for (i = 11; i > 0; i--)  
01184             if (d0[i] <= doy)  
01185                 break;  
01186         *mon = i + 1;  
01187         *day = doy - d0[i] + 1;  
01188     }  
01189 }
```


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

Convert geolocation to Cartesian coordinates.

Definition at line 1193 of file [libtrac.c](#).

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

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

Get meteo data for given time step.

Definition at line 1207 of file [libtrac.c](#).

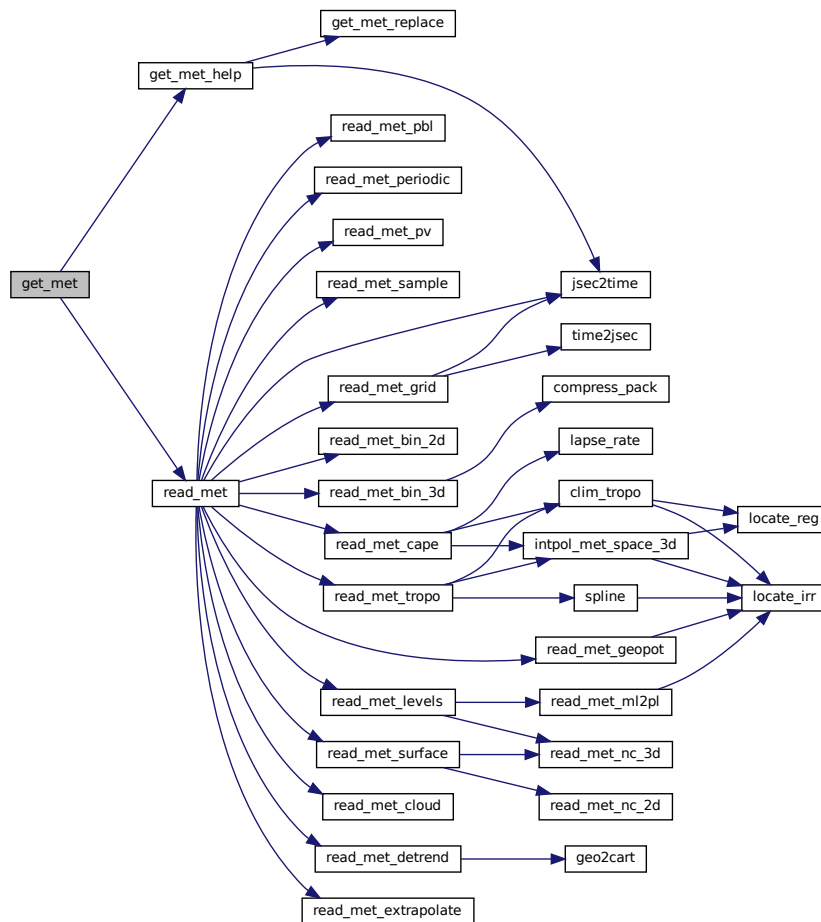
```
01212     {
01213
01214     static int init;
01215
01216     met_t *mets;
01217
01218     char cachefile[LEN], cmd[2 * LEN], filename[LEN];
01219
01220     /* Set timer... */
01221     SELECT_TIMER("GET_MET", "INPUT", NVTX_READ);
01222
01223     /* Init... */
01224     if (t == ctl->t_start || !init) {
01225         init = 1;
01226
01227         /* Read meteo data... */
01228         get_met_help(ctl, t + (ctl->direction == -1 ? -1 : 0), -1,
01229                     ctl->metbase, ctl->dt_met, filename);
01230         if (!read_met(filename, ctl, clim, *met0))
01231             ERRMSG("Cannot open file!");
01232
01233         get_met_help(ctl, t + (ctl->direction == 1 ? 1 : 0), 1,
01234                     ctl->metbase, ctl->dt_met, filename);
01235         if (!read_met(filename, ctl, clim, *met1))
01236             ERRMSG("Cannot open file!");
01237
01238         /* Update GPU... */
01239         #ifdef _OPENACC
01240             met_t *met0up = *met0;
01241             met_t *met1up = *met1;
01242         #endif
01243         #pragma acc update device(met0up[:1],met1up[:1]) async(5)
01244         #else
01245         #pragma acc update device(met0up[:1],met1up[:1])
01246         #endif
01247         #endif
01248
01249         /* Caching... */
01250         if (ctl->met_cache && t != ctl->t_stop) {
01251             get_met_help(ctl, t + 1.1 * ctl->dt_met * ctl->direction,
01252                         ctl->direction, ctl->metbase, ctl->dt_met, cachefile);
01253             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01254             LOG(1, "Caching: %s", cachefile);
01255             if (system(cmd) != 0)
01256                 WARN("Caching command failed!");
01257         }
01258     }
```

```

01258     }
01259
01260     /* Read new data for forward trajectories... */
01261     if (t > (*met1)->time) {
01262
01263         /* Pointer swap... */
01264         mets = *met1;
01265         *met1 = *met0;
01266         *met0 = mets;
01267
01268         /* Read new meteo data... */
01269         get_met_help(ctl, t, 1, ctl->metbase, ctl->dt_met, filename);
01270         if (!read_met(filename, ctl, clim, *met1))
01271             ERRMSG("Cannot open file!");
01272
01273         /* Update GPU... */
01274         #ifdef _OPENACC
01275             met_t *metlup = *met1;
01276         #ifdef ASYNCIO
01277             #pragma acc update device(metlup[:1]) async(5)
01278         #else
01279             #pragma acc update device(metlup[:1])
01280         #endif
01281         #endif
01282
01283         /* Caching... */
01284         if (ctl->met_cache && t != ctl->t_stop) {
01285             get_met_help(ctl, t + ctl->dt_met, 1, ctl->metbase, ctl->dt_met,
01286                         cachefile);
01287             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01288             LOG(1, "Caching: %s", cachefile);
01289             if (system(cmd) != 0)
01290                 WARN("Caching command failed!");
01291         }
01292     }
01293     /* Read new data for backward trajectories... */
01294     if (t < (*met0)->time) {
01295
01296         /* Pointer swap... */
01297         mets = *met1;
01298         *met1 = *met0;
01299         *met0 = mets;
01300
01301         /* Read new meteo data... */
01302         get_met_help(ctl, t, -1, ctl->metbase, ctl->dt_met, filename);
01303         if (!read_met(filename, ctl, clim, *met0))
01304             ERRMSG("Cannot open file!");
01305
01306         /* Update GPU... */
01307         #ifdef _OPENACC
01308             met_t *met0up = *met0;
01309         #ifdef ASYNCIO
01310             #pragma acc update device(met0up[:1]) async(5)
01311         #else
01312             #pragma acc update device(met0up[:1])
01313         #endif
01314         #endif
01315
01316         /* Caching... */
01317         if (ctl->met_cache && t != ctl->t_stop) {
01318             get_met_help(ctl, t - ctl->dt_met, -1, ctl->metbase, ctl->dt_met,
01319                         cachefile);
01320             sprintf(cmd, "cat %s > /dev/null &", cachefile);
01321             LOG(1, "Caching: %s", cachefile);
01322             if (system(cmd) != 0)
01323                 WARN("Caching command failed!");
01324         }
01325     }
01326     /* Check that grids are consistent... */
01327     if ((*met0)->nx != 0 && (*met1)->nx != 0) {
01328         if ((*met0)->nx != (*met1)->nx
01329             || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01330             ERRMSG("Meteo grid dimensions do not match!");
01331         for (int ix = 0; ix < (*met0)->nx; ix++)
01332             if (fabs((*met0)->lon[ix] - (*met1)->lon[ix]) > 0.001)
01333                 ERRMSG("Meteo grid longitudes do not match!");
01334         for (int iy = 0; iy < (*met0)->ny; iy++)
01335             if (fabs((*met0)->lat[iy] - (*met1)->lat[iy]) > 0.001)
01336                 ERRMSG("Meteo grid latitudes do not match!");
01337         for (int ip = 0; ip < (*met0)->np; ip++)
01338             if (fabs((*met0)->p[ip] - (*met1)->p[ip]) > 0.001)
01339                 ERRMSG("Meteo grid pressure levels do not match!");
01340     }
01341 }

```

Here is the call graph for this function:



5.23.3.20 get_met_help() void get_met_help (

```

    ctl_t * ctl,
    double t,
    int direct,
    char * metbase,
    double dt_met,
    char * filename )

```

Get meteo data for time step.

Definition at line 1345 of file libtrac.c.

```

01351 {
01352
01353     char repl[LEN];
01354
01355     double t6, r;
01356
01357     int year, mon, day, hour, min, sec;
01358
01359     /* Round time to fixed intervals... */
01360     if (direct == -1)
01361         t6 = floor(t / dt_met) * dt_met;

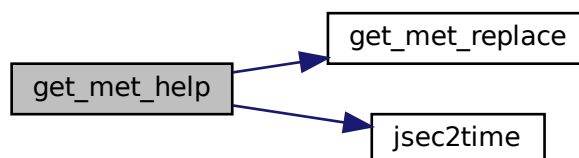
```

```

01362     else
01363         t6 = ceil(t / dt_met) * dt_met;
01364
01365     /* Decode time... */
01366     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01367
01368     /* Set filename of MPTRAC meteo files... */
01369     if (ctl->clams_met_data == 0) {
01370         if (ctl->met_type == 0)
01371             sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01372         else if (ctl->met_type == 1)
01373             sprintf(filename, "%s_YYYY_MM_DD_HH.bin", metbase);
01374         else if (ctl->met_type == 2)
01375             sprintf(filename, "%s_YYYY_MM_DD_HH.pck", metbase);
01376         else if (ctl->met_type == 3)
01377             sprintf(filename, "%s_YYYY_MM_DD_HH.zfp", metbase);
01378         else if (ctl->met_type == 4)
01379             sprintf(filename, "%s_YYYY_MM_DD_HH.zstd", metbase);
01380         sprintf(repl, "%d", year);
01381         get_met_replace(filename, "YYYY", repl);
01382         sprintf(repl, "%02d", mon);
01383         get_met_replace(filename, "MM", repl);
01384         sprintf(repl, "%02d", day);
01385         get_met_replace(filename, "DD", repl);
01386         sprintf(repl, "%02d", hour);
01387         get_met_replace(filename, "HH", repl);
01388     }
01389
01390     /* Set filename of CLaMS meteo files... */
01391     else {
01392         sprintf(filename, "%s_YYMMDDHH.nc", metbase);
01393         sprintf(repl, "%d", year);
01394         get_met_replace(filename, "YYYY", repl);
01395         sprintf(repl, "%d", year % 100);
01396         get_met_replace(filename, "YY", repl);
01397         sprintf(repl, "%02d", mon);
01398         get_met_replace(filename, "MM", repl);
01399         sprintf(repl, "%02d", day);
01400         get_met_replace(filename, "DD", repl);
01401         sprintf(repl, "%02d", hour);
01402         get_met_replace(filename, "HH", repl);
01403     }
01404 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1408 of file libtrac.c.

```

01411     {

```

```

01412
01413     char buffer[LEN];
01414
01415     /* Iterate... */
01416     for (int i = 0; i < 3; i++) {
01417
01418         /* Replace sub-string... */
01419         char *ch;
01420         if (!(ch = strstr(orig, search)))
01421             return;
01422         strncpy(buffer, orig, (size_t) (ch - orig));
01423         buffer[ch - orig] = 0;
01424         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01425         orig[0] = 0;
01426         strcpy(orig, buffer);
01427     }
01428 }

```

5.23.3.23 intpol_met_space_3d() void intpol_met_space_3d (

```

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

```

Spatial interpolation of meteo data.

Definition at line 1432 of file libtrac.c.

```

01441     {
01442
01443     /* Initialize interpolation... */
01444     if (init) {
01445
01446         /* Check longitude... */
01447         if (met->lon[met->nx - 1] > 180 && lon < 0)
01448             lon += 360;
01449
01450         /* Get interpolation indices... */
01451         ci[0] = locate_irr(met->p, met->np, p);
01452         ci[1] = locate_reg(met->lon, met->nx, lon);
01453         ci[2] = locate_reg(met->lat, met->ny, lat);
01454
01455         /* Get interpolation weights... */
01456         cw[0] = (met->p[ci[0] + 1] - p)
01457             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01458         cw[1] = (met->lon[ci[1] + 1] - lon)
01459             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01460         cw[2] = (met->lat[ci[2] + 1] - lat)
01461             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01462     }
01463
01464     /* Interpolate vertically... */
01465     double aux00 =
01466         cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01467         + array[ci[1]][ci[2]][ci[0] + 1];
01468     double aux01 =
01469         cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01470             array[ci[1]][ci[2] + 1][ci[0] + 1])
01471         + array[ci[1]][ci[2] + 1][ci[0] + 1];
01472     double aux10 =
01473         cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01474             array[ci[1] + 1][ci[2]][ci[0] + 1])
01475         + array[ci[1] + 1][ci[2]][ci[0] + 1];
01476     double aux11 =
01477         cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01478             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01479         + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01480
01481     /* Interpolate horizontally... */
01482     aux00 = cw[2] * (aux00 - aux01) + aux01;
01483     aux11 = cw[2] * (aux10 - aux11) + aux11;

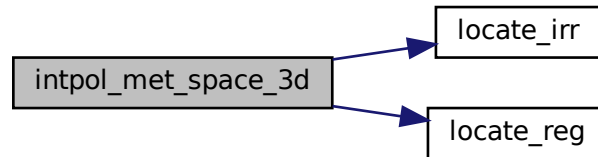
```

```

01484  *var = cw[1] * (aux00 - aux11) + aux11;
01485 }

```

Here is the call graph for this function:



5.23.3.23 intpol_met_space_2d() void intpol_met_space_2d (

```

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

```

Spatial interpolation of meteo data.

Definition at line 1489 of file libtrac.c.

```

01497     {
01498
01499     /* Initialize interpolation... */
01500     if (init) {
01501
01502         /* Check longitude... */
01503         if (met->lon[met->nx - 1] > 180 && lon < 0)
01504             lon += 360;
01505
01506         /* Get interpolation indices... */
01507         ci[1] = locate_reg(met->lon, met->nx, lon);
01508         ci[2] = locate_reg(met->lat, met->ny, lat);
01509
01510         /* Get interpolation weights... */
01511         cw[1] = (met->lon[ci[1] + 1] - lon)
01512             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01513         cw[2] = (met->lat[ci[2] + 1] - lat)
01514             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01515     }
01516
01517     /* Set variables... */
01518     double aux00 = array[ci[1]][ci[2]];
01519     double aux01 = array[ci[1]][ci[2] + 1];
01520     double aux10 = array[ci[1] + 1][ci[2]];
01521     double aux11 = array[ci[1] + 1][ci[2] + 1];
01522
01523     /* Interpolate horizontally... */
01524     if (isfinite(aux00) && isfinite(aux01)
01525         && isfinite(aux10) && isfinite(aux11)) {
01526         aux00 = cw[2] * (aux00 - aux01) + aux01;
01527         aux11 = cw[2] * (aux10 - aux11) + aux11;
01528         *var = cw[1] * (aux00 - aux11) + aux11;
01529     } else {
01530         if (cw[2] < 0.5) {
01531             if (cw[1] < 0.5)

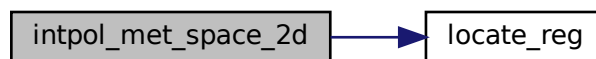
```

```

01532     *var = aux11;
01533     else
01534     *var = aux01;
01535 } else {
01536     if (cw[1] < 0.5)
01537     *var = aux10;
01538     else
01539     *var = aux00;
01540 }
01541 }
01542 }

```

Here is the call graph for this function:



5.23.3.24 intpol_met_time_3d() void intpol_met_time_3d (

```

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

```

Spatial interpolation of meteo data.

Temporal interpolation of meteo data.

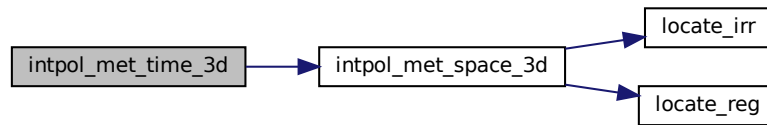
Definition at line 1649 of file libtrac.c.

```

01661     {
01662
01663     double var0, var1, wt;
01664
01665     /* Spatial interpolation... */
01666     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01667     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, 0);
01668
01669     /* Get weighting factor... */
01670     wt = (met1->time - ts) / (met1->time - met0->time);
01671
01672     /* Interpolate... */
01673     *var = wt * (var0 - var1) + var1;
01674 }

```

Here is the call graph for this function:



5.23.3.25 intpol_met_time_2d() void intpol_met_time_2d (

```

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

```

Temporal interpolation of meteo data.

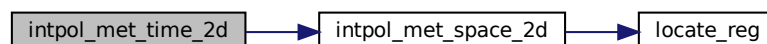
Definition at line 1678 of file libtrac.c.

```

01689     {
01690
01691     double var0, var1, wt;
01692
01693     /* Spatial interpolation... */
01694     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01695     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, 0);
01696
01697     /* Get weighting factor... */
01698     wt = (met1->time - ts) / (met1->time - met0->time);
01699
01700     /* Interpolate... */
01701     if (isfinite(var0) && isfinite(var1))
01702         *var = wt * (var0 - var1) + var1;
01703     else if (wt < 0.5)
01704         *var = var1;
01705     else
01706         *var = var0;
01707 }

```

Here is the call graph for this function:



5.23.3.26 jsec2time() void jsec2time (

```

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

```

Temporal interpolation of meteo data.

Convert seconds to date.

Definition at line 1742 of file libtrac.c.

```

01750     {
01751
01752     struct tm t0, *t1;
01753
01754     t0.tm_year = 100;
01755     t0.tm_mon = 0;
01756     t0.tm_mday = 1;
01757     t0.tm_hour = 0;
01758     t0.tm_min = 0;
01759     t0.tm_sec = 0;
01760
01761     time_t jsec0 = (time_t) jsec + timegm(&t0);
01762     t1 = gmtime(&jsec0);
01763
01764     *year = t1->tm_year + 1900;
01765     *mon = t1->tm_mon + 1;
01766     *day = t1->tm_mday;
01767     *hour = t1->tm_hour;
01768     *min = t1->tm_min;
01769     *sec = t1->tm_sec;
01770     *remain = jsec - floor(jsec);
01771 }

```

5.23.3.27 lapse_rate() double lapse_rate (

```

    double t,
    double h2o )

```

Calculate moist adiabatic lapse rate.

Definition at line 1775 of file libtrac.c.

```

01777     {
01778
01779     /*
01780      Calculate moist adiabatic lapse rate [K/km] from temperature [K]
01781      and water vapor volume mixing ratio [1].
01782
01783      Reference: https://en.wikipedia.org/wiki/Lapse\_rate
01784     */
01785
01786     const double a = RA * SQR(t), r = SH(h2o) / (1. - SH(h2o));
01787
01788     return 1e3 * G0 * (a + LV * r * t) / (CPD * a + SQR(LV) * r * EPS);
01789 }

```

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

Find array index for irregular grid.

Definition at line 1793 of file libtrac.c.

```
01796     {
01797
01798     int ilo = 0;
01799     int ihi = n - 1;
01800     int i = (ihi + ilo) » 1;
01801
01802     if (xx[i] < xx[i + 1])
01803         while (ihi > ilo + 1) {
01804             i = (ihi + ilo) » 1;
01805             if (xx[i] > x)
01806                 ihi = i;
01807             else
01808                 ilo = i;
01809         } else
01810             while (ihi > ilo + 1) {
01811                 i = (ihi + ilo) » 1;
01812                 if (xx[i] <= x)
01813                     ihi = i;
01814                 else
01815                     ilo = i;
01816             }
01817
01818     return ilo;
01819 }
```

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

Find array index for regular grid.

Definition at line 1823 of file libtrac.c.

```
01826     {
01827
01828     /* Calculate index... */
01829     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01830
01831     /* Check range... */
01832     if (i < 0)
01833         return 0;
01834     else if (i > n - 2)
01835         return n - 2;
01836     else
01837         return i;
01838 }
```

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

Calculate NAT existence temperature.

Definition at line 1842 of file libtrac.c.

```
01845     {
01846
01847     /* Check water vapor vmr... */
```

```

01848     h2o = GSL_MAX(h2o, 0.1e-6);
01849
01850     /* Calculate T_NAT... */
01851     double p_hno3 = hno3 * p / 1.333224;
01852     double p_h2o = h2o * p / 1.333224;
01853     double a = 0.009179 - 0.00088 * log10(p_h2o);
01854     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01855     double c = -11397.0 / a;
01856     double tnat = (-b + sqrt(b * b - 4. * c)) / 2.;
01857     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01858     if (x2 > 0)
01859         tnat = x2;
01860
01861     return tnat;
01862 }

```

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

Parallel quicksort.

Definition at line 1866 of file libtrac.c.

```

01870     {
01871
01872     if (low < high) {
01873         int pi = quicksort_partition(arr, brr, low, high);
01874
01875     #pragma omp task firstprivate(arr,brr,low,pi)
01876         {
01877             quicksort(arr, brr, low, pi - 1);
01878         }
01879
01880     // #pragma omp task firstprivate(arr,brr,high,pi)
01881         {
01882             quicksort(arr, brr, pi + 1, high);
01883         }
01884     }
01885 }

```

Here is the call graph for this function:



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

Partition function for quicksort.

Definition at line 1889 of file libtrac.c.

```

01893     {
01894
01895     double pivot = arr[high];
01896     int i = (low - 1);
01897
01898     for (int j = low; j <= high - 1; j++)
01899         if (arr[j] <= pivot) {
01900             i++;
01901             SWAP(arr[i], arr[j], double);
01902             SWAP(brr[i], brr[j], int);
01903         }
01904     SWAP(arr[high], arr[i + 1], double);
01905     SWAP(brr[high], brr[i + 1], int);
01906
01907     return (i + 1);
01908 }
```

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

Read atmospheric data.

Definition at line 1912 of file libtrac.c.

```

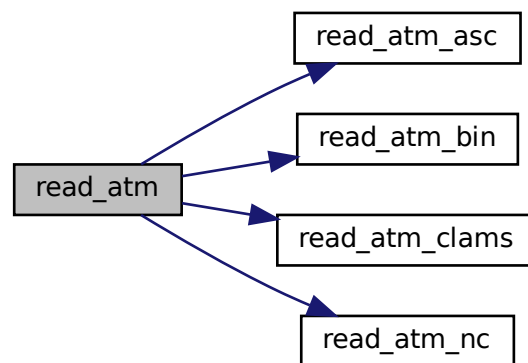
01915     {
01916
01917     int result;
01918
01919     /* Set timer... */
01920     SELECT_TIMER("READ_ATM", "INPUT", NVTX_READ);
01921
01922     /* Init... */
01923     atm->np = 0;
01924
01925     /* Write info... */
01926     LOG(1, "Read atmospheric data: %s", filename);
01927
01928     /* Read ASCII data... */
01929     if (ctl->atm_type == 0)
01930         result = read_atm_asc(filename, ctl, atm);
01931
01932     /* Read binary data... */
01933     else if (ctl->atm_type == 1)
01934         result = read_atm_bin(filename, ctl, atm);
01935
01936     /* Read netCDF data... */
01937     else if (ctl->atm_type == 2)
01938         result = read_atm_nc(filename, ctl, atm);
01939
01940     /* Read CLaMS data... */
01941     else if (ctl->atm_type == 3)
01942         result = read_atm_clams(filename, ctl, atm);
01943
01944     /* Error... */
01945     else
01946         ERRMSG("Atmospheric data type not supported!");
01947
01948     /* Check result... */
01949     if (result != 1)
01950         return 0;
01951
01952     /* Check number of air parcels... */
01953     if (atm->np < 1)
01954         ERRMSG("Can not read any data!");
01955
01956     /* Write info... */
01957     double mini, maxi;
01958     LOG(2, "Number of particles: %d", atm->np);
01959     gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
01960     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
01961     gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
01962     LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
01963     LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
01964     gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
01965     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
01966     gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
```

```

01967 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
01968 for (int iq = 0; iq < ctl->nq; iq++) {
01969     char msg[LEN];
01970     sprintf(msg, "Quantity %s range: %s ... %s %s",
01971             ctl->qnt_name[iq], ctl->qnt_format[iq],
01972             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
01973     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
01974     LOG(2, msg, mini, maxi);
01975 }
01976
01977 /* Return success... */
01978 return 1;
01979 }

```

Here is the call graph for this function:



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

Read atmospheric data in ASCII format.

Definition at line 1983 of file `libtrac.c`.

```

01986     {
01987
01988     FILE *in;
01989
01990     /* Open file... */
01991     if (!(in = fopen(filename, "r"))) {
01992         WARN("Cannot open file!");
01993         return 0;
01994     }
01995
01996     /* Read line... */
01997     char line[LEN];
01998     while (fgets(line, LEN, in)) {
01999
02000         /* Read data... */
02001         char *tok;
02002         TOK(line, tok, "%lg", atm->time[atm->np]);
02003         TOK(NULL, tok, "%lg", atm->p[atm->np]);
02004         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02005         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02006         for (int iq = 0; iq < ctl->nq; iq++)
02007             TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);

```

```

02008
02009     /* Convert altitude to pressure... */
02010     atm->p[atm->np] = P(atm->p[atm->np]);
02011
02012     /* Increment data point counter... */
02013     if ((++atm->np) > NP)
02014         ERRMSG("Too many data points!");
02015 }
02016
02017 /* Close file... */
02018 fclose(in);
02019
02020 /* Return success... */
02021 return 1;
02022 }

```

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

Read atmospheric data in binary format.

Definition at line 2026 of file libtrac.c.

```

02029     {
02030
02031     FILE *in;
02032
02033     /* Open file... */
02034     if (!(in = fopen(filename, "r")))
02035         return 0;
02036
02037     /* Check version of binary data... */
02038     int version;
02039     FREAD(&version, int,
02040          1,
02041          in);
02042     if (version != 100)
02043         ERRMSG("Wrong version of binary data!");
02044
02045     /* Read data... */
02046     FREAD(&atm->np, int,
02047          1,
02048          in);
02049     FREAD(atm->time, double,
02050          (size_t) atm->np,
02051          in);
02052     FREAD(atm->p, double,
02053          (size_t) atm->np,
02054          in);
02055     FREAD(atm->lon, double,
02056          (size_t) atm->np,
02057          in);
02058     FREAD(atm->lat, double,
02059          (size_t) atm->np,
02060          in);
02061     for (int iq = 0; iq < ctl->nq; iq++)
02062         FREAD(atm->q[iq], double,
02063              (size_t) atm->np,
02064              in);
02065
02066     /* Read final flag... */
02067     int final;
02068     FREAD(&final, int,
02069          1,
02070          in);
02071     if (final != 999)
02072         ERRMSG("Error while reading binary data!");
02073
02074     /* Close file... */
02075     fclose(in);
02076
02077     /* Return success... */
02078     return 1;
02079 }

```

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

Read atmospheric data in CLaMS format.

Definition at line 2083 of file libtrac.c.

```
02086     {
02087
02088     int ncid, varid;
02089
02090     /* Open file... */
02091     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02092         return 0;
02093
02094     /* Get dimensions... */
02095     NC_INQ_DIM("NPARTS", &atm->np, 1, NP);
02096
02097     /* Get time... */
02098     if (nc_inq_varid(ncid, "TIME_INIT", &varid) == NC_NOERR) {
02099         NC(nc_get_var_double(ncid, varid, atm->time));
02100     } else {
02101         WARN("TIME_INIT not found use time instead!");
02102         double time_init;
02103         NC_GET_DOUBLE("time", &time_init, 1);
02104         for (int ip = 0; ip < atm->np; ip++) {
02105             atm->time[ip] = time_init;
02106         }
02107     }
02108
02109     /* Read zeta coordinate, pressure is optional... */
02110     if (ctl->vert_coord_ap == 1) {
02111         NC_GET_DOUBLE("ZETA", atm->zeta, 1);
02112         NC_GET_DOUBLE("PRESS", atm->p, 0);
02113     }
02114
02115     /* Read pressure, zeta coordinate is optional... */
02116     else {
02117         NC_GET_DOUBLE("PRESS", atm->p, 1);
02118         NC_GET_DOUBLE("ZETA", atm->zeta, 0);
02119     }
02120
02121     /* Read longitude and latitude... */
02122     NC_GET_DOUBLE("LON", atm->lon, 1);
02123     NC_GET_DOUBLE("LAT", atm->lat, 1);
02124
02125     /* Close file... */
02126     NC(nc_close(ncid));
02127
02128     /* Return success... */
02129     return 1;
02130 }
```

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

Read atmospheric data in netCDF format.

Definition at line 2134 of file libtrac.c.

```
02137     {
02138
02139     int ncid, varid;
02140
02141     /* Open file... */
02142     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02143         return 0;
02144
02145     /* Get dimensions... */
02146     NC_INQ_DIM("obs", &atm->np, 1, NP);
02147
02148     /* Read geolocations... */
02149     NC_GET_DOUBLE("time", atm->time, 1);
```

```

02150 NC_GET_DOUBLE("press", atm->p, 1);
02151 NC_GET_DOUBLE("lon", atm->lon, 1);
02152 NC_GET_DOUBLE("lat", atm->lat, 1);
02153
02154 /* Read variables... */
02155 for (int iq = 0; iq < ctl->nq; iq++)
02156     NC_GET_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
02157
02158 /* Close file... */
02159 NC(nc_close(ncid));
02160
02161 /* Return success... */
02162 return 1;
02163 }

```

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

Read climatological data.

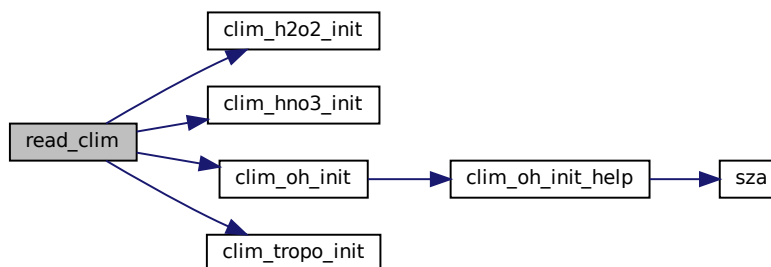
Definition at line 2167 of file libtrac.c.

```

02169 {
02170
02171     /* Set timer... */
02172     SELECT_TIMER("READ_CLIM", "INPUT", NVTX_READ);
02173
02174     /* Init tropopause climatology... */
02175     clim_tropo_init(clim);
02176
02177     /* Init HNO3 climatology... */
02178     clim_hno3_init(clim);
02179
02180     /* Read OH climatology... */
02181     if (ctl->clim_oh_filename[0] != '-')
02182         clim_oh_init(ctl, clim);
02183
02184     /* Read H2O2 climatology... */
02185     if (ctl->clim_h2o2_filename[0] != '-')
02186         clim_h2o2_init(ctl, clim);
02187 }

```

Here is the call graph for this function:




```

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

```

Read control parameters.

Definition at line 2191 of file [libtrac.c](#).

```

02195     {
02196
02197         /* Set timer... */
02198         SELECT_TIMER("READ_CTL", "INPUT", NVTX_READ);
02199
02200         /* Write info... */
02201         LOG(1, "\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02202             "(executable: %s | version: %s | compiled: %s, %s)\n",
02203             argv[0], VERSION, __DATE__, __TIME__);
02204
02205         /* Initialize quantity indices... */
02206         ctl->qnt_idx = -1;
02207         ctl->qnt_ens = -1;
02208         ctl->qnt_stat = -1;
02209         ctl->qnt_m = -1;
02210         ctl->qnt_vmr = -1;
02211         ctl->qnt_rp = -1;
02212         ctl->qnt_rhop = -1;
02213         ctl->qnt_ps = -1;
02214         ctl->qnt_ts = -1;
02215         ctl->qnt_zs = -1;
02216         ctl->qnt_us = -1;
02217         ctl->qnt_vs = -1;
02218         ctl->qnt_pbl = -1;
02219         ctl->qnt_pt = -1;
02220         ctl->qnt_tt = -1;
02221         ctl->qnt_zt = -1;
02222         ctl->qnt_h2ot = -1;
02223         ctl->qnt_z = -1;
02224         ctl->qnt_p = -1;
02225         ctl->qnt_t = -1;
02226         ctl->qnt_rho = -1;
02227         ctl->qnt_u = -1;
02228         ctl->qnt_v = -1;
02229         ctl->qnt_w = -1;
02230         ctl->qnt_h2o = -1;
02231         ctl->qnt_o3 = -1;
02232         ctl->qnt_lwc = -1;
02233         ctl->qnt_iwc = -1;
02234         ctl->qnt_pct = -1;
02235         ctl->qnt_pcb = -1;
02236         ctl->qnt_cl = -1;
02237         ctl->qnt_plcl = -1;
02238         ctl->qnt_plfc = -1;
02239         ctl->qnt_pel = -1;
02240         ctl->qnt_cape = -1;
02241         ctl->qnt_cin = -1;
02242         ctl->qnt_hno3 = -1;
02243         ctl->qnt_oh = -1;
02244         ctl->qnt_vmrimpl = -1;
02245         ctl->qnt_mloss_oh = -1;
02246         ctl->qnt_mloss_h2o2 = -1;
02247         ctl->qnt_mloss_wet = -1;
02248         ctl->qnt_mloss_dry = -1;
02249         ctl->qnt_mloss_decay = -1;
02250         ctl->qnt_psat = -1;
02251         ctl->qnt_psice = -1;
02252         ctl->qnt_pw = -1;
02253         ctl->qnt_sh = -1;
02254         ctl->qnt_rh = -1;
02255         ctl->qnt_rhice = -1;
02256         ctl->qnt_theta = -1;
02257         ctl->qnt_zeta = -1;
02258         ctl->qnt_tvirt = -1;
02259         ctl->qnt_lapse = -1;
02260         ctl->qnt_vh = -1;
02261         ctl->qnt_vz = -1;
02262         ctl->qnt_pv = -1;
02263         ctl->qnt_tdew = -1;
02264         ctl->qnt_tice = -1;
02265         ctl->qnt_tsts = -1;
02266         ctl->qnt_tnat = -1;
02267
02268         /* Read quantities... */

```

```

02269     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02270     if (ctl->nq > NQ)
02271         ERRMSG("Too many quantities!");
02272     for (int iq = 0; iq < ctl->nq; iq++) {
02273
02274         /* Read quantity name and format... */
02275         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02276         scan_ctl(filename, argc, argv, "QNT_LONGNAME", iq, ctl->qnt_name[iq],
02277             ctl->qnt_longname[iq]);
02278         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02279             ctl->qnt_format[iq]);
02280
02281         /* Try to identify quantity... */
02282         SET_QNT(qnt_idx, "idx", "particle index", "-")
02283         SET_QNT(qnt_ens, "ens", "ensemble index", "-")
02284         SET_QNT(qnt_stat, "stat", "station flag", "-")
02285         SET_QNT(qnt_m, "m", "mass", "kg")
02286         SET_QNT(qnt_vmr, "vmr", "volume mixing ratio", "ppv")
02287         SET_QNT(qnt_rp, "rp", "particle radius", "microns")
02288         SET_QNT(qnt_rhop, "rhop", "particle density", "kg/m^3")
02289         SET_QNT(qnt_ps, "ps", "surface pressure", "hPa")
02290         SET_QNT(qnt_ts, "ts", "surface temperature", "K")
02291         SET_QNT(qnt_zs, "zs", "surface height", "km")
02292         SET_QNT(qnt_us, "us", "surface zonal wind", "m/s")
02293         SET_QNT(qnt_vs, "vs", "surface meridional wind", "m/s")
02294         SET_QNT(qnt_pbl, "pbl", "planetary boundary layer", "hPa")
02295         SET_QNT(qnt_pt, "pt", "tropopause pressure", "hPa")
02296         SET_QNT(qnt_tt, "tt", "tropopause temperature", "K")
02297         SET_QNT(qnt_zt, "zt", "tropopause geopotential height", "km")
02298         SET_QNT(qnt_h2ot, "h2ot", "tropopause water vapor", "ppv")
02299         SET_QNT(qnt_z, "z", "geopotential height", "km")
02300         SET_QNT(qnt_p, "p", "pressure", "hPa")
02301         SET_QNT(qnt_t, "t", "temperature", "K")
02302         SET_QNT(qnt_rho, "rho", "air density", "kg/m^3")
02303         SET_QNT(qnt_u, "u", "zonal wind", "m/s")
02304         SET_QNT(qnt_v, "v", "meridional wind", "m/s")
02305         SET_QNT(qnt_w, "w", "vertical velocity", "hPa/s")
02306         SET_QNT(qnt_h2o, "h2o", "water vapor", "ppv")
02307         SET_QNT(qnt_o3, "o3", "ozone", "ppv")
02308         SET_QNT(qnt_lwc, "lwc", "cloud ice water content", "kg/kg")
02309         SET_QNT(qnt_lwc, "lwc", "cloud liquid water content", "kg/kg")
02310         SET_QNT(qnt_pct, "pct", "cloud top pressure", "hPa")
02311         SET_QNT(qnt_pcb, "pcb", "cloud bottom pressure", "hPa")
02312         SET_QNT(qnt_cl, "cl", "total column cloud water", "kg/m^2")
02313         SET_QNT(qnt_plcl, "plcl", "lifted condensation level", "hPa")
02314         SET_QNT(qnt_plfc, "plfc", "level of free convection", "hPa")
02315         SET_QNT(qnt_pel, "pel", "equilibrium level", "hPa")
02316         SET_QNT(qnt_cape, "cape", "convective available potential energy",
02317             "J/kg")
02318         SET_QNT(qnt_cin, "cin", "convective inhibition", "J/kg")
02319         SET_QNT(qnt_hno3, "hno3", "nitric acid", "ppv")
02320         SET_QNT(qnt_oh, "oh", "hydroxyl radical", "molec/cm^3")
02321         SET_QNT(qnt_vmrimpl, "vmrimpl", "volume mixing ratio (implicit)", "ppv")
02322         SET_QNT(qnt_mloss_oh, "mloss_oh", "mass loss due to OH chemistry", "kg")
02323         SET_QNT(qnt_mloss_h2o2, "mloss_h2o2", "mass loss due to H2O2 chemistry",
02324             "kg")
02325         SET_QNT(qnt_mloss_wet, "mloss_wet", "mass loss due to wet deposition",
02326             "kg")
02327         SET_QNT(qnt_mloss_dry, "mloss_dry", "mass loss due to dry deposition",
02328             "kg")
02329         SET_QNT(qnt_mloss_decay, "mloss_decay",
02330             "mass loss due to exponential decay", "kg")
02331         SET_QNT(qnt_psat, "psat", "saturation pressure over water", "hPa")
02332         SET_QNT(qnt_psice, "psice", "saturation pressure over ice", "hPa")
02333         SET_QNT(qnt_pw, "pw", "partial water vapor pressure", "hPa")
02334         SET_QNT(qnt_sh, "sh", "specific humidity", "kg/kg")
02335         SET_QNT(qnt_rh, "rh", "relative humidity", "%")
02336         SET_QNT(qnt_rhice, "rhice", "relative humidity over ice", "%")
02337         SET_QNT(qnt_theta, "theta", "potential temperature", "K")
02338         SET_QNT(qnt_zeta, "zeta", "zeta coordinate", "K")
02339         SET_QNT(qnt_tvirt, "tvirt", "virtual temperature", "K")
02340         SET_QNT(qnt_lapse, "lapse", "temperature lapse rate", "K/km")
02341         SET_QNT(qnt_vh, "vh", "horizontal velocity", "m/s")
02342         SET_QNT(qnt_vz, "vz", "vertical velocity", "m/s")
02343         SET_QNT(qnt_pv, "pv", "potential vorticity", "PVU")
02344         SET_QNT(qnt_tdew, "tdew", "dew point temperature", "K")
02345         SET_QNT(qnt_tice, "tice", "frost point temperature", "K")
02346         SET_QNT(qnt_tsts, "tsts", "STS existence temperature", "K")
02347         SET_QNT(qnt_tnat, "tnat", "NAT existence temperature", "K")
02348         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02349     }
02350
02351     /* netCDF I/O parameters... */
02352     ctl->chunkszhint =
02353         (size_t) scan_ctl(filename, argc, argv, "CHUNKSZHINT", -1, "163840000",
02354             NULL);
02355     ctl->read_mode =

```

```

02356     (int) scan_ctl(filename, argc, argv, "READMODE", -1, "0", NULL);
02357
02358 /* Vertical coordinates and velocities... */
02359 ctl->vert_coord_ap =
02360     (int) scan_ctl(filename, argc, argv, "VERT_COORD_AP", -1, "0", NULL);
02361 ctl->vert_coord_met =
02362     (int) scan_ctl(filename, argc, argv, "VERT_COORD_MET", -1, "0", NULL);
02363 ctl->vert_vel =
02364     (int) scan_ctl(filename, argc, argv, "VERT_VEL", -1, "0", NULL);
02365 ctl->clams_met_data =
02366     (int) scan_ctl(filename, argc, argv, "CLAMS_MET_DATA", -1, "0", NULL);
02367
02368 /* Time steps of simulation... */
02369 ctl->direction =
02370     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02371 if (ctl->direction != -1 && ctl->direction != 1)
02372     ERRMSG("Set DIRECTION to -1 or 1!");
02373 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02374 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "180", NULL);
02375
02376 /* Meteo data... */
02377 scan_ctl(filename, argc, argv, "METBASE", -1, "-", ctl->metbase);
02378 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "3600", NULL);
02379 ctl->met_type =
02380     (int) scan_ctl(filename, argc, argv, "MET_TYPE", -1, "0", NULL);
02381 ctl->met_nc_scale =
02382     (int) scan_ctl(filename, argc, argv, "MET_NC_SCALE", -1, "1", NULL);
02383 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02384 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02385 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02386 if (ctl->met_dx < 1 || ctl->met_dy < 1 || ctl->met_dp < 1)
02387     ERRMSG("MET_DX, MET_DY, and MET_DP need to be greater than zero!");
02388 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02389 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02390 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02391 if (ctl->met_sx < 1 || ctl->met_sy < 1 || ctl->met_sp < 1)
02392     ERRMSG("MET_SX, MET_SY, and MET_SP need to be greater than zero!");
02393 ctl->met_detrend =
02394     scan_ctl(filename, argc, argv, "MET_DETREND", -1, "-999", NULL);
02395 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02396 if (ctl->met_np > EP)
02397     ERRMSG("Too many levels!");
02398 for (int ip = 0; ip < ctl->met_np; ip++)
02399     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02400 ctl->met_geopot_sx =
02401     (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SX", -1, "-1", NULL);
02402 ctl->met_geopot_sy =
02403     (int) scan_ctl(filename, argc, argv, "MET_GEOPOT_SY", -1, "-1", NULL);
02404 ctl->met_relhun =
02405     (int) scan_ctl(filename, argc, argv, "MET_RELHUM", -1, "0", NULL);
02406 ctl->met_tropo =
02407     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02408 if (ctl->met_tropo < 0 || ctl->met_tropo > 5)
02409     ERRMSG("Set MET_TROPO = 0 ... 5!");
02410 ctl->met_tropo_lapse =
02411     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE", -1, "2.0", NULL);
02412 ctl->met_tropo_nlev =
02413     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV", -1, "20", NULL);
02414 ctl->met_tropo_lapse_sep =
02415     scan_ctl(filename, argc, argv, "MET_TROPO_LAPSE_SEP", -1, "3.0", NULL);
02416 ctl->met_tropo_nlev_sep =
02417     (int) scan_ctl(filename, argc, argv, "MET_TROPO_NLEV_SEP", -1, "10",
02418     NULL);
02419 ctl->met_tropo_pv =
02420     scan_ctl(filename, argc, argv, "MET_TROPO_PV", -1, "3.5", NULL);
02421 ctl->met_tropo_theta =
02422     scan_ctl(filename, argc, argv, "MET_TROPO_THETA", -1, "380", NULL);
02423 ctl->met_tropo_spline =
02424     (int) scan_ctl(filename, argc, argv, "MET_TROPO_SPLINE", -1, "1", NULL);
02425 ctl->met_cloud =
02426     (int) scan_ctl(filename, argc, argv, "MET_CLOUD", -1, "1", NULL);
02427 if (ctl->met_cloud < 0 || ctl->met_cloud > 3)
02428     ERRMSG("Set MET_CLOUD = 0 ... 3!");
02429 ctl->met_cloud_min =
02430     scan_ctl(filename, argc, argv, "MET_CLOUD_MIN", -1, "0", NULL);
02431 ctl->met_dt_out =
02432     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02433 ctl->met_cache =
02434     (int) scan_ctl(filename, argc, argv, "MET_CACHE", -1, "0", NULL);
02435
02436 /* Sorting... */
02437 ctl->sort_dt = scan_ctl(filename, argc, argv, "SORT_DT", -1, "-999", NULL);
02438
02439 /* Isosurface parameters... */
02440 ctl->isosurf =
02441     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02442 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);

```

```

02443
02444 /* Advection parameters... */
02445 ctl->advect = (int) scan_ctl(filename, argc, argv, "ADVECT", -1, "2", NULL);
02446 if (!(ctl->advect == 1 || ctl->advect == 2 || ctl->advect == 4))
02447     ERRMSG("Set ADVECT to 1, 2, or 4!");
02448 ctl->reflect =
02449     (int) scan_ctl(filename, argc, argv, "REFLECT", -1, "0", NULL);
02450
02451 /* Diffusion parameters... */
02452 ctl->turb_dx_trop =
02453     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02454 ctl->turb_dx_strat =
02455     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02456 ctl->turb_dz_trop =
02457     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02458 ctl->turb_dz_strat =
02459     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02460 ctl->turb_mesox =
02461     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02462 ctl->turb_mesoz =
02463     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02464
02465 /* Convection... */
02466 ctl->conv_cape =
02467     scan_ctl(filename, argc, argv, "CONV_CAPE", -1, "-999", NULL);
02468 ctl->conv_cin =
02469     scan_ctl(filename, argc, argv, "CONV_CIN", -1, "-999", NULL);
02470 ctl->conv_dt = scan_ctl(filename, argc, argv, "CONV_DT", -1, "-999", NULL);
02471 ctl->conv_mix =
02472     (int) scan_ctl(filename, argc, argv, "CONV_MIX", -1, "1", NULL);
02473 ctl->conv_mix_bot =
02474     (int) scan_ctl(filename, argc, argv, "CONV_MIX_BOT", -1, "1", NULL);
02475 ctl->conv_mix_top =
02476     (int) scan_ctl(filename, argc, argv, "CONV_MIX_TOP", -1, "1", NULL);
02477
02478 /* Boundary conditions... */
02479 ctl->bound_mass =
02480     scan_ctl(filename, argc, argv, "BOUND_MASS", -1, "-999", NULL);
02481 ctl->bound_mass_trend =
02482     scan_ctl(filename, argc, argv, "BOUND_MASS_TREND", -1, "0", NULL);
02483 ctl->bound_vmr =
02484     scan_ctl(filename, argc, argv, "BOUND_VMR", -1, "-999", NULL);
02485 ctl->bound_vmr_trend =
02486     scan_ctl(filename, argc, argv, "BOUND_VMR_TREND", -1, "0", NULL);
02487 ctl->bound_lat0 =
02488     scan_ctl(filename, argc, argv, "BOUND_LAT0", -1, "-90", NULL);
02489 ctl->bound_lat1 =
02490     scan_ctl(filename, argc, argv, "BOUND_LAT1", -1, "90", NULL);
02491 ctl->bound_p0 =
02492     scan_ctl(filename, argc, argv, "BOUND_P0", -1, "1e10", NULL);
02493 ctl->bound_p1 =
02494     scan_ctl(filename, argc, argv, "BOUND_P1", -1, "-1e10", NULL);
02495 ctl->bound_dps =
02496     scan_ctl(filename, argc, argv, "BOUND_DPS", -1, "-999", NULL);
02497 ctl->bound_dzs =
02498     scan_ctl(filename, argc, argv, "BOUND_DZS", -1, "-999", NULL);
02499 ctl->bound_zetas =
02500     scan_ctl(filename, argc, argv, "BOUND_ZETAS", -1, "-999", NULL);
02501 ctl->bound_pbl =
02502     (int) scan_ctl(filename, argc, argv, "BOUND_PBL", -1, "0", NULL);
02503
02504 /* Species parameters... */
02505 scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02506 if (strcasecmp(ctl->species, "CF2Cl2") == 0) {
02507     ctl->molmass = 120.907;
02508     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3e-5;
02509     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3500.0;
02510 } else if (strcasecmp(ctl->species, "CFC13") == 0) {
02511     ctl->molmass = 137.359;
02512     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.1e-4;
02513     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3300.0;
02514 } else if (strcasecmp(ctl->species, "CH4") == 0) {
02515     ctl->molmass = 16.043;
02516     ctl->oh_chem_reaction = 2;
02517     ctl->oh_chem[0] = 2.45e-12;
02518     ctl->oh_chem[1] = 1775;
02519     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.4e-5;
02520     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02521 } else if (strcasecmp(ctl->species, "CO") == 0) {
02522     ctl->molmass = 28.01;
02523     ctl->oh_chem_reaction = 3;
02524     ctl->oh_chem[0] = 6.9e-33;
02525     ctl->oh_chem[1] = 2.1;
02526     ctl->oh_chem[2] = 1.1e-12;
02527     ctl->oh_chem[3] = -1.3;
02528     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 9.7e-6;
02529     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1300.0;

```

```
02530 } else if (strcasecmp(ctl->species, "CO2") == 0) {
02531     ctl->molmass = 44.009;
02532     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 3.3e-4;
02533     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02534 } else if (strcasecmp(ctl->species, "H2O") == 0) {
02535     ctl->molmass = 18.01528;
02536 } else if (strcasecmp(ctl->species, "N2O") == 0) {
02537     ctl->molmass = 44.013;
02538     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-4;
02539     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2600.;
02540 } else if (strcasecmp(ctl->species, "NH3") == 0) {
02541     ctl->molmass = 17.031;
02542     ctl->oh_chem_reaction = 2;
02543     ctl->oh_chem[0] = 1.7e-12;
02544     ctl->oh_chem[1] = 710;
02545     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 5.9e-1;
02546     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 4200.0;
02547 } else if (strcasecmp(ctl->species, "HNO3") == 0) {
02548     ctl->molmass = 63.012;
02549     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.1e3;
02550     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 8700.0;
02551 } else if (strcasecmp(ctl->species, "NO") == 0) {
02552     ctl->molmass = 30.006;
02553     ctl->oh_chem_reaction = 3;
02554     ctl->oh_chem[0] = 7.1e-31;
02555     ctl->oh_chem[1] = 2.6;
02556     ctl->oh_chem[2] = 3.6e-11;
02557     ctl->oh_chem[3] = 0.1;
02558     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.9e-5;
02559     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 1600.0;
02560 } else if (strcasecmp(ctl->species, "NO2") == 0) {
02561     ctl->molmass = 46.005;
02562     ctl->oh_chem_reaction = 3;
02563     ctl->oh_chem[0] = 1.8e-30;
02564     ctl->oh_chem[1] = 3.0;
02565     ctl->oh_chem[2] = 2.8e-11;
02566     ctl->oh_chem[3] = 0.0;
02567     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.2e-4;
02568     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2400.0;
02569 } else if (strcasecmp(ctl->species, "O3") == 0) {
02570     ctl->molmass = 47.997;
02571     ctl->oh_chem_reaction = 2;
02572     ctl->oh_chem[0] = 1.7e-12;
02573     ctl->oh_chem[1] = 940;
02574     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1e-4;
02575     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2800.0;
02576 } else if (strcasecmp(ctl->species, "SF6") == 0) {
02577     ctl->molmass = 146.048;
02578     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 2.4e-6;
02579     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 3100.0;
02580 } else if (strcasecmp(ctl->species, "SO2") == 0) {
02581     ctl->molmass = 64.066;
02582     ctl->oh_chem_reaction = 3;
02583     ctl->oh_chem[0] = 2.9e-31;
02584     ctl->oh_chem[1] = 4.1;
02585     ctl->oh_chem[2] = 1.7e-12;
02586     ctl->oh_chem[3] = -0.2;
02587     ctl->wet_depo_ic_h[0] = ctl->wet_depo_bc_h[0] = 1.3e-2;
02588     ctl->wet_depo_ic_h[1] = ctl->wet_depo_bc_h[1] = 2900.0;
02589 } else {
02590     ctl->molmass =
02591         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02592     ctl->oh_chem_reaction =
02593         (int) scan_ctl(filename, argc, argv, "OH_CHEM_REACTION", -1, "0", NULL);
02594     ctl->h2o2_chem_reaction =
02595         (int) scan_ctl(filename, argc, argv, "H2O2_CHEM_REACTION", -1, "0",
02596             NULL);
02597     for (int ip = 0; ip < 4; ip++)
02598         ctl->oh_chem[ip] =
02599             scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02600     ctl->dry_depo_vdep =
02601         scan_ctl(filename, argc, argv, "DRY_DEPO_VDEP", -1, "0", NULL);
02602     ctl->dry_depo_dp =
02603         scan_ctl(filename, argc, argv, "DRY_DEPO_DP", -1, "30", NULL);
02604     ctl->wet_depo_ic_a =
02605         scan_ctl(filename, argc, argv, "WET_DEPO_IC_A", -1, "0", NULL);
02606     ctl->wet_depo_ic_b =
02607         scan_ctl(filename, argc, argv, "WET_DEPO_IC_B", -1, "0", NULL);
02608     ctl->wet_depo_bc_a =
02609         scan_ctl(filename, argc, argv, "WET_DEPO_BC_A", -1, "0", NULL);
02610     ctl->wet_depo_bc_b =
02611         scan_ctl(filename, argc, argv, "WET_DEPO_BC_B", -1, "0", NULL);
02612     for (int ip = 0; ip < 3; ip++)
02613         ctl->wet_depo_ic_h[ip] =
02614             scan_ctl(filename, argc, argv, "WET_DEPO_IC_H", ip, "0", NULL);
02615     for (int ip = 0; ip < 1; ip++)
02616         ctl->wet_depo_bc_h[ip] =
```

```

02617         scan_ctl(filename, argc, argv, "WET_DEPO_BC_H", ip, "0", NULL);
02618     }
02619
02620     /* Wet deposition... */
02621     ctl->wet_depo_pre[0] =
02622         scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 0, "0.5", NULL);
02623     ctl->wet_depo_pre[1] =
02624         scan_ctl(filename, argc, argv, "WET_DEPO_PRE", 1, "0.36", NULL);
02625     ctl->wet_depo_ic_ret_ratio =
02626         scan_ctl(filename, argc, argv, "WET_DEPO_IC_RET_RATIO", -1, "1", NULL);
02627     ctl->wet_depo_bc_ret_ratio =
02628         scan_ctl(filename, argc, argv, "WET_DEPO_BC_RET_RATIO", -1, "1", NULL);
02629
02630     /* OH chemistry... */
02631     ctl->oh_chem_beta =
02632         scan_ctl(filename, argc, argv, "OH_CHEM_BETA", -1, "0", NULL);
02633     scan_ctl(filename, argc, argv, "CLIM_OH_FILENAME", -1,
02634         ".../data/clams_radical_species.nc", ctl->clim_oh_filename);
02635
02636     /* H2O2 chemistry... */
02637     ctl->h2o2_chem_cc =
02638         scan_ctl(filename, argc, argv, "H2O2_CHEM_CC", -1, "1", NULL);
02639     scan_ctl(filename, argc, argv, "CLIM_H2O2_FILENAME", -1,
02640         ".../data/cams_H2O2.nc", ctl->clim_h2o2_filename);
02641
02642     /* Chemistry grid... */
02643     ctl->chemgrid_z0 =
02644         scan_ctl(filename, argc, argv, "CHEMGRID_Z0", -1, "0", NULL);
02645     ctl->chemgrid_z1 =
02646         scan_ctl(filename, argc, argv, "CHEMGRID_Z1", -1, "100", NULL);
02647     ctl->chemgrid_nz =
02648         (int) scan_ctl(filename, argc, argv, "CHEMGRID_NZ", -1, "1", NULL);
02649     ctl->chemgrid_lon0 =
02650         scan_ctl(filename, argc, argv, "CHEMGRID_LON0", -1, "-180", NULL);
02651     ctl->chemgrid_lon1 =
02652         scan_ctl(filename, argc, argv, "CHEMGRID_LON1", -1, "180", NULL);
02653     ctl->chemgrid_nx =
02654         (int) scan_ctl(filename, argc, argv, "CHEMGRID_NX", -1, "360", NULL);
02655     ctl->chemgrid_lat0 =
02656         scan_ctl(filename, argc, argv, "CHEMGRID_LAT0", -1, "-90", NULL);
02657     ctl->chemgrid_lat1 =
02658         scan_ctl(filename, argc, argv, "CHEMGRID_LAT1", -1, "90", NULL);
02659     ctl->chemgrid_ny =
02660         (int) scan_ctl(filename, argc, argv, "CHEMGRID_NY", -1, "180", NULL);
02661
02662     /* Exponential decay... */
02663     ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02664     ctl->tdec_strat =
02665         scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02666
02667     /* PSC analysis... */
02668     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02669     ctl->psc_hno3 =
02670         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02671
02672     /* Output of atmospheric data... */
02673     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->atm_basename);
02674     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02675     ctl->atm_dt_out =
02676         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02677     ctl->atm_filter =
02678         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02679     ctl->atm_stride =
02680         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02681     ctl->atm_type =
02682         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02683
02684     /* Output of CSI data... */
02685     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->csi_basename);
02686     ctl->csi_dt_out =
02687         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02688     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->csi_obsfile);
02689     ctl->csi_obsmin =
02690         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02691     ctl->csi_modmin =
02692         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02693     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02694     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02695     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02696     ctl->csi_lon0 =
02697         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02698     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02699     ctl->csi_nx =
02700         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02701     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02702     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02703     ctl->csi_ny =

```

```

02704     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02705
02706     /* Output of ensemble data... */
02707     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->ens_basename);
02708     ctl->ens_dt_out =
02709         scan_ctl(filename, argc, argv, "ENS_DT_OUT", -1, "86400", NULL);
02710
02711     /* Output of grid data... */
02712     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02713         ctl->grid_basename);
02714     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->grid_gpfile);
02715     ctl->grid_dt_out =
02716         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02717     ctl->grid_sparse =
02718         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02719     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02720     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02721     ctl->grid_nz =
02722         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02723     ctl->grid_lon0 =
02724         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02725     ctl->grid_lon1 =
02726         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02727     ctl->grid_nx =
02728         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02729     ctl->grid_lat0 =
02730         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02731     ctl->grid_lat1 =
02732         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02733     ctl->grid_ny =
02734         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02735     ctl->grid_type =
02736         (int) scan_ctl(filename, argc, argv, "GRID_TYPE", -1, "0", NULL);
02737
02738     /* Output of profile data... */
02739     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02740         ctl->prof_basename);
02741     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->prof_obsfile);
02742     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02743     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02744     ctl->prof_nz =
02745         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02746     ctl->prof_lon0 =
02747         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02748     ctl->prof_lon1 =
02749         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02750     ctl->prof_nx =
02751         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02752     ctl->prof_lat0 =
02753         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02754     ctl->prof_lat1 =
02755         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02756     ctl->prof_ny =
02757         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02758
02759     /* Output of sample data... */
02760     scan_ctl(filename, argc, argv, "SAMPLE_BASENAME", -1, "-",
02761         ctl->sample_basename);
02762     scan_ctl(filename, argc, argv, "SAMPLE_OBSFILE", -1, "-",
02763         ctl->sample_obsfile);
02764     ctl->sample_dx =
02765         scan_ctl(filename, argc, argv, "SAMPLE_DX", -1, "50", NULL);
02766     ctl->sample_dz =
02767         scan_ctl(filename, argc, argv, "SAMPLE_DZ", -1, "-999", NULL);
02768
02769     /* Output of station data... */
02770     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02771         ctl->stat_basename);
02772     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02773     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02774     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02775     ctl->stat_t0 =
02776         scan_ctl(filename, argc, argv, "STAT_T0", -1, "-1e100", NULL);
02777     ctl->stat_t1 = scan_ctl(filename, argc, argv, "STAT_T1", -1, "1e100", NULL);
02778 }

```

Here is the call graph for this function:



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

Read meteo data file.

Definition at line 2782 of file libtrac.c.

```

02786     {
02787
02788     /* Write info... */
02789     LOG(1, "Read meteo data: %s", filename);
02790
02791     /* Read netCDF data... */
02792     if (ctl->met_type == 0) {
02793
02794         int ncid;
02795
02796         /* Open netCDF file... */
02797         if (nc_open(filename, ctl->read_mode, &ctl->chunkszhint, &ncid) !=
02798             NC_NOERR) {
02799             WARN("Cannot open file!");
02800             return 0;
02801         }
02802
02803         /* Read coordinates of meteo data... */
02804         read_met_grid(filename, ncid, ctl, met);
02805
02806         /* Read meteo data on vertical levels... */
02807         read_met_levels(ncid, ctl, met);
02808
02809         /* Extrapolate data for lower boundary... */
02810         read_met_extrapolate(met);
02811
02812         /* Read surface data... */
02813         read_met_surface(ncid, met, ctl);
02814
02815         /* Create periodic boundary conditions... */
02816         read_met_periodic(met);
02817
02818         /* Downsampling... */
02819         read_met_sample(ctl, met);
02820
02821         /* Calculate geopotential heights... */
02822         read_met_geopot(ctl, met);
02823
02824         /* Calculate potential vorticity... */
02825         read_met_pv(met);
02826
02827         /* Calculate boundary layer data... */
02828         read_met_pbl(met);
02829
02830         /* Calculate tropopause data... */
02831         read_met_tropo(ctl, clim, met);
02832
02833         /* Calculate cloud properties... */
02834         read_met_cloud(ctl, met);
  
```



```

02835
02836     /* Calculate convective available potential energy... */
02837     read_met_cape(clim, met);
02838
02839     /* Detrending... */
02840     read_met_detrend(ctl, met);
02841
02842     /* Close file... */
02843     NC(nc_close(ncid));
02844 }
02845
02846 /* Read binary data... */
02847 else if (ctl->met_type >= 1 && ctl->met_type <= 4) {
02848
02849     FILE *in;
02850
02851     double r;
02852
02853     int year, mon, day, hour, min, sec;
02854
02855     /* Set timer... */
02856     SELECT_TIMER("READ_MET_BIN", "INPUT", NVTX_READ);
02857
02858     /* Open file... */
02859     if (!(in = fopen(filename, "r"))) {
02860         WARN("Cannot open file!");
02861         return 0;
02862     }
02863
02864     /* Check type of binary data... */
02865     int met_type;
02866     FREAD(&met_type, int,
02867          1,
02868          in);
02869     if (met_type != ctl->met_type)
02870         ERRMSG("Wrong MET_TYPE of binary data!");
02871
02872     /* Check version of binary data... */
02873     int version;
02874     FREAD(&version, int,
02875          1,
02876          in);
02877     if (version != 100)
02878         ERRMSG("Wrong version of binary data!");
02879
02880     /* Read time... */
02881     FREAD(&met->time, double,
02882          1,
02883          in);
02884     jsec2time(met->time, &year, &mon, &day, &hour, &min, &sec, &r);
02885     LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
02886         met->time, year, mon, day, hour, min);
02887     if (year < 1900 || year > 2100 || mon < 1 || mon > 12
02888         || day < 1 || day > 31 || hour < 0 || hour > 23)
02889         ERRMSG("Error while reading time!");
02890
02891     /* Read dimensions... */
02892     FREAD(&met->nx, int,
02893          1,
02894          in);
02895     LOG(2, "Number of longitudes: %d", met->nx);
02896     if (met->nx < 2 || met->nx > EX)
02897         ERRMSG("Number of longitudes out of range!");
02898
02899     FREAD(&met->ny, int,
02900          1,
02901          in);
02902     LOG(2, "Number of latitudes: %d", met->ny);
02903     if (met->ny < 2 || met->ny > EY)
02904         ERRMSG("Number of latitudes out of range!");
02905
02906     FREAD(&met->np, int,
02907          1,
02908          in);
02909     LOG(2, "Number of levels: %d", met->np);
02910     if (met->np < 2 || met->np > EP)
02911         ERRMSG("Number of levels out of range!");
02912
02913     /* Read grid... */
02914     FREAD(met->lon, double,
02915          (size_t) met->nx,
02916          in);
02917     LOG(2, "Longitudes: %g, %g ... %g deg",
02918         met->lon[0], met->lon[1], met->lon[met->nx - 1]);
02919
02920     FREAD(met->lat, double,
02921          (size_t) met->ny,

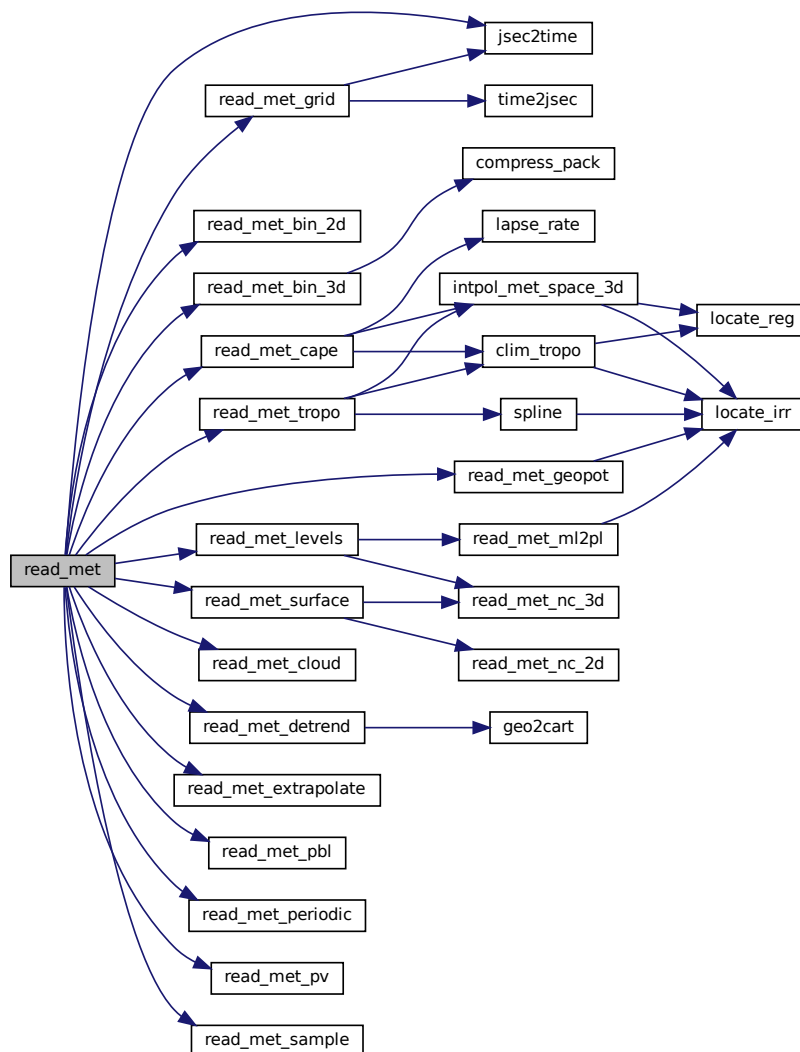
```

```

02922         in);
02923     LOG(2, "Latitudes: %g, %g ... %g deg",
02924         met->lat[0], met->lat[1], met->lat[met->ny - 1]);
02925
02926     FREAD(met->p, double,
02927         (size_t) met->np,
02928         in);
02929     LOG(2, "Altitude levels: %g, %g ... %g km",
02930         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));
02931     LOG(2, "Pressure levels: %g, %g ... %g hPa",
02932         met->p[0], met->p[1], met->p[met->np - 1]);
02933
02934     /* Read surface data... */
02935     read_met_bin_2d(in, met, met->ps, "PS");
02936     read_met_bin_2d(in, met, met->ts, "TS");
02937     read_met_bin_2d(in, met, met->zs, "ZS");
02938     read_met_bin_2d(in, met, met->us, "US");
02939     read_met_bin_2d(in, met, met->vs, "VS");
02940     read_met_bin_2d(in, met, met->pbl, "PBL");
02941     read_met_bin_2d(in, met, met->pt, "PT");
02942     read_met_bin_2d(in, met, met->tt, "TT");
02943     read_met_bin_2d(in, met, met->zt, "ZT");
02944     read_met_bin_2d(in, met, met->h2ot, "H2OT");
02945     read_met_bin_2d(in, met, met->pct, "PCT");
02946     read_met_bin_2d(in, met, met->pcb, "PCB");
02947     read_met_bin_2d(in, met, met->cl, "CL");
02948     read_met_bin_2d(in, met, met->plcl, "PLCL");
02949     read_met_bin_2d(in, met, met->plfc, "PLFC");
02950     read_met_bin_2d(in, met, met->pel, "PEL");
02951     read_met_bin_2d(in, met, met->cape, "CAPE");
02952     read_met_bin_2d(in, met, met->cin, "CIN");
02953
02954     /* Read level data... */
02955     read_met_bin_3d(in, ctl, met, met->z, "Z", 0, 0.5);
02956     read_met_bin_3d(in, ctl, met, met->t, "T", 0, 5.0);
02957     read_met_bin_3d(in, ctl, met, met->u, "U", 8, 0);
02958     read_met_bin_3d(in, ctl, met, met->v, "V", 8, 0);
02959     read_met_bin_3d(in, ctl, met, met->w, "W", 8, 0);
02960     read_met_bin_3d(in, ctl, met, met->pv, "PV", 8, 0);
02961     read_met_bin_3d(in, ctl, met, met->h2o, "H2O", 8, 0);
02962     read_met_bin_3d(in, ctl, met, met->o3, "O3", 8, 0);
02963     read_met_bin_3d(in, ctl, met, met->lwc, "LWC", 8, 0);
02964     read_met_bin_3d(in, ctl, met, met->iwc, "IWC", 8, 0);
02965
02966     /* Read final flag... */
02967     int final;
02968     FREAD(&final, int,
02969         1,
02970         in);
02971     if (final != 999)
02972         ERRMSG("Error while reading binary data!");
02973
02974     /* Close file... */
02975     fclose(in);
02976 }
02977
02978 /* Not implemented... */
02979 else
02980     ERRMSG("MET_TYPE not implemented!");
02981
02982 /* Copy wind data to cache... */
02983 #ifdef UVW
02984 #pragma omp parallel for default(shared) collapse(2)
02985     for (int ix = 0; ix < met->nx; ix++)
02986         for (int iy = 0; iy < met->ny; iy++)
02987             for (int ip = 0; ip < met->np; ip++) {
02988                 met->uvw[ix][iy][ip][0] = met->u[ix][iy][ip];
02989                 met->uvw[ix][iy][ip][1] = met->v[ix][iy][ip];
02990                 met->uvw[ix][iy][ip][2] = met->w[ix][iy][ip];
02991             }
02992 #endif
02993
02994 /* Return success... */
02995 return 1;
02996 }

```

Here is the call graph for this function:



5.23.3.41 read_met_bin_2d() void read_met_bin_2d (

```

FILE * out,
met_t * met,
float var[EX][EY],
char * varname )

```

Read 2-D meteo variable.

Definition at line 3000 of file libtrac.c.

```

03004 {
03005
03006     float *help;
03007
03008     /* Allocate... */
03009     ALLOC(help, float,
03010           EX * EY);

```

```

03011
03012 /* Read uncompressed... */
03013 LOG(2, "Read 2-D variable: %s (uncompressed)", varname);
03014 FREAD(help, float,
03015       (size_t) (met->nx * met->ny),
03016       in);
03017
03018 /* Copy data... */
03019 for (int ix = 0; ix < met->nx; ix++)
03020     for (int iy = 0; iy < met->ny; iy++)
03021         var[ix][iy] = help[ARRAY_2D(ix, iy, met->ny)];
03022
03023 /* Free... */
03024 free(help);
03025 }

```

5.23.3.42 read_met_bin_3d() void read_met_bin_3d (

```

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

```

Read 3-D meteo variable.

Definition at line 3029 of file libtrac.c.

```

03036 {
03037
03038     float *help;
03039
03040     /* Allocate... */
03041     ALLOC(help, float,
03042           EX * EY * EP);
03043
03044     /* Read uncompressed data... */
03045     if (ctl->met_type == 1) {
03046         LOG(2, "Read 3-D variable: %s (uncompressed)", varname);
03047         FREAD(help, float,
03048               (size_t) (met->nx * met->ny * met->np),
03049               in);
03050     }
03051
03052     /* Read packed data... */
03053     else if (ctl->met_type == 2)
03054         compress_pack(varname, help, (size_t) (met->ny * met->nx),
03055                       (size_t) met->np, 1, in);
03056
03057     /* Read zfp data... */
03058     else if (ctl->met_type == 3) {
03059 #ifdef ZFP
03060         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
03061                     tolerance, 1, in);
03062     #else
03063         ERRMSG("zfp compression not supported!");
03064         LOG(3, "%d %g", precision, tolerance);
03065     #endif
03066     }
03067
03068     /* Read zstd data... */
03069     else if (ctl->met_type == 4) {
03070 #ifdef ZSTD
03071         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 1,
03072                     in);
03073     #else
03074         ERRMSG("zstd compression not supported!");
03075     #endif
03076     }
03077
03078     /* Copy data... */
03079 #pragma omp parallel for default(shared) collapse(2)
03080     for (int ix = 0; ix < met->nx; ix++)
03081         for (int iy = 0; iy < met->ny; iy++)
03082             for (int ip = 0; ip < met->np; ip++)
03083                 var[ix][iy][ip] = help[ARRAY_3D(ix, iy, met->ny, ip, met->np)];

```

```

03084
03085  /* Free... */
03086  free(help);
03087 }

```

Here is the call graph for this function:



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

Calculate convective available potential energy.

Definition at line 3091 of file libtrac.c.

```

03093     {
03094
03095     /* Set timer... */
03096     SELECT_TIMER("READ_MET_CAPE", "METPROC", NVTX_READ);
03097     LOG(2, "Calculate CAPE...");
03098
03099     /* Vertical spacing (about 100 m)... */
03100     const double pfac = 1.01439, dz0 = RI / MA / G0 * log(pfac);
03101
03102     /* Loop over columns... */
03103     #pragma omp parallel for default(shared) collapse(2)
03104     for (int ix = 0; ix < met->nx; ix++)
03105         for (int iy = 0; iy < met->ny; iy++) {
03106
03107             /* Get potential temperature and water vapor vmr at lowest 50 hPa... */
03108             int n = 0;
03109             double h2o = 0, t, theta = 0;
03110             double pbot = GSL_MIN(met->ps[ix][iy], met->p[0]);
03111             double ptop = pbot - 50.;
03112             for (int ip = 0; ip < met->np; ip++) {
03113                 if (met->p[ip] <= pbot) {
03114                     theta += THETA(met->p[ip], met->t[ix][iy][ip]);
03115                     h2o += met->h2o[ix][iy][ip];
03116                     n++;
03117                 }
03118                 if (met->p[ip] < ptop && n > 0)
03119                     break;
03120             }
03121             theta /= n;
03122             h2o /= n;
03123
03124             /* Cannot compute anything if water vapor is missing... */
03125             met->plcl[ix][iy] = GSL_NAN;
03126             met->plfc[ix][iy] = GSL_NAN;
03127             met->pel[ix][iy] = GSL_NAN;
03128             met->cape[ix][iy] = GSL_NAN;
03129             met->cin[ix][iy] = GSL_NAN;
03130             if (h2o <= 0)
03131                 continue;
03132
03133             /* Find lifted condensation level (LCL)... */
03134             ptop = P(20.);
03135             pbot = met->ps[ix][iy];
03136             do {
03137                 met->plcl[ix][iy] = (float) (0.5 * (pbot + ptop));
03138                 t = theta / pow(1000. / met->plcl[ix][iy], 0.286);
03139                 if (RH(met->plcl[ix][iy], t, h2o) > 100.)

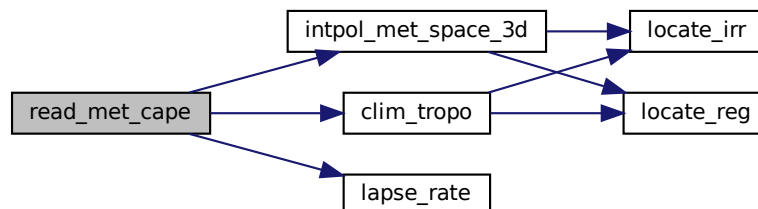
```

```

03140     ptop = met->plcl[ix][iy];
03141     else
03142         pbot = met->plcl[ix][iy];
03143     } while (pbot - ptop > 0.1);
03144
03145     /* Calculate CIN up to LCL... */
03146     INTPOL_INIT;
03147     double dcaper, dz, h2o_env, t_env;
03148     double p = met->ps[ix][iy];
03149     met->cape[ix][iy] = met->cin[ix][iy] = 0;
03150     do {
03151         dz = dz0 * TVIRT(t, h2o);
03152         p /= pfac;
03153         t = theta / pow(1000. / p, 0.286);
03154         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03155                             &t_env, ci, cw, 1);
03156         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03157                             &h2o_env, ci, cw, 0);
03158         dcaper = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03159                 TVIRT(t_env, h2o_env) * dz;
03160         if (dcaper < 0)
03161             met->cin[ix][iy] += fabsf((float) dcaper);
03162     } while (p > met->plcl[ix][iy]);
03163
03164     /* Calculate level of free convection (LFC), equilibrium level (EL),
03165        and convective available potential energy (CAPE)... */
03166     dcaper = 0;
03167     p = met->plcl[ix][iy];
03168     t = theta / pow(1000. / p, 0.286);
03169     ptop = 0.75 * clim_tropo(clim, met->time, met->lat[iy]);
03170     do {
03171         dz = dz0 * TVIRT(t, h2o);
03172         p /= pfac;
03173         t -= lapse_rate(t, h2o) * dz;
03174         double psat = PSAT(t);
03175         h2o = psat / (p - (1. - EPS) * psat);
03176         intpol_met_space_3d(met, met->t, p, met->lon[ix], met->lat[iy],
03177                             &t_env, ci, cw, 1);
03178         intpol_met_space_3d(met, met->h2o, p, met->lon[ix], met->lat[iy],
03179                             &h2o_env, ci, cw, 0);
03180         double dcaper_old = dcaper;
03181         dcaper = 1e3 * G0 * (TVIRT(t, h2o) - TVIRT(t_env, h2o_env)) /
03182                 TVIRT(t_env, h2o_env) * dz;
03183         if (dcaper > 0) {
03184             met->cape[ix][iy] += (float) dcaper;
03185             if (!isfinite(met->plfc[ix][iy]))
03186                 met->plfc[ix][iy] = (float) p;
03187         } else if (dcaper_old > 0)
03188             met->pel[ix][iy] = (float) p;
03189         if (dcaper < 0 && !isfinite(met->plfc[ix][iy]))
03190             met->cin[ix][iy] += fabsf((float) dcaper);
03191     } while (p > ptop);
03192
03193     /* Check results... */
03194     if (!isfinite(met->plfc[ix][iy]))
03195         met->cin[ix][iy] = GSL_NAN;
03196 }
03197 }

```

Here is the call graph for this function:



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

Calculate cloud properties.

Definition at line 3201 of file libtrac.c.

```
03203     {
03204
03205     /* Set timer... */
03206     SELECT_TIMER("READ_MET_CLOUD", "METPROC", NVTX_READ);
03207     LOG(2, "Calculate cloud data...");
03208
03209     /* Loop over columns... */
03210     #pragma omp parallel for default(shared) collapse(2)
03211     for (int ix = 0; ix < met->nx; ix++)
03212         for (int iy = 0; iy < met->ny; iy++) {
03213
03214         /* Init... */
03215         met->pct[ix][iy] = GSL_NAN;
03216         met->pcb[ix][iy] = GSL_NAN;
03217         met->cl[ix][iy] = 0;
03218
03219         /* Loop over pressure levels... */
03220         for (int ip = 0; ip < met->np - 1; ip++) {
03221
03222             /* Check pressure... */
03223             if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
03224                 continue;
03225
03226             /* Check ice water and liquid water content... */
03227             if (met->iwc[ix][iy][ip] > ctl->met_cloud_min
03228                 || met->lwc[ix][iy][ip] > ctl->met_cloud_min) {
03229
03230                 /* Get cloud top pressure ... */
03231                 met->pct[ix][iy]
03232                     = (float) (0.5 * (met->p[ip] + (float) met->p[ip + 1]));
03233
03234                 /* Get cloud bottom pressure ... */
03235                 if (!isfinite(met->pcb[ix][iy]))
03236                     met->pcb[ix][iy]
03237                         = (float) (0.5 * (met->p[ip] + met->p[GSL_MAX(ip - 1, 0)]));
03238             }
03239
03240             /* Get cloud water... */
03241             met->cl[ix][iy] += (float)
03242                 (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
03243                     + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
03244                 * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
03245         }
03246     }
03247 }
```

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

Apply detrending method to temperature and winds.

Definition at line 3251 of file libtrac.c.

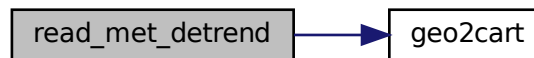
```
03253     {
03254
03255     met_t *help;
03256
03257     /* Check parameters... */
03258     if (ctl->met_detrend <= 0)
03259         return;
03260
03261     /* Set timer... */
03262     SELECT_TIMER("READ_MET_DETREND", "METPROC", NVTX_READ);
03263     LOG(2, "Detrend meteo data...");
03264
03265     /* Allocate... */
03266     ALLOC(help, met_t, 1);
03267
03268     /* Calculate standard deviation... */
```

```

03269 double sigma = ctl->met_detrend / 2.355;
03270 double tssq = 2. * SQR(sigma);
03271
03272 /* Calculate box size in latitude... */
03273 int sy = (int) (3. * DY2DEG(sigma) / fabs(met->lat[1] - met->lat[0]));
03274 sy = GSL_MIN(GSL_MAX(1, sy), met->ny / 2);
03275
03276 /* Calculate background... */
03277 #pragma omp parallel for default(shared) collapse(2)
03278 for (int ix = 0; ix < met->nx; ix++) {
03279     for (int iy = 0; iy < met->ny; iy++) {
03280
03281         /* Calculate Cartesian coordinates... */
03282         double x0[3];
03283         geo2cart(0.0, met->lon[ix], met->lat[iy], x0);
03284
03285         /* Calculate box size in longitude... */
03286         int sx =
03287             (int) (3. * DX2DEG(sigma, met->lat[iy]) /
03288                 fabs(met->lon[1] - met->lon[0]));
03289         sx = GSL_MIN(GSL_MAX(1, sx), met->nx / 2);
03290
03291         /* Init... */
03292         float wsum = 0;
03293         for (int ip = 0; ip < met->np; ip++) {
03294             help->t[ix][iy][ip] = 0;
03295             help->u[ix][iy][ip] = 0;
03296             help->v[ix][iy][ip] = 0;
03297             help->w[ix][iy][ip] = 0;
03298         }
03299
03300         /* Loop over neighboring grid points... */
03301         for (int ix2 = ix - sx; ix2 <= ix + sx; ix2++) {
03302             int ix3 = ix2;
03303             if (ix3 < 0)
03304                 ix3 += met->nx;
03305             else if (ix3 >= met->nx)
03306                 ix3 -= met->nx;
03307             for (int iy2 = GSL_MAX(iy - sy, 0);
03308                 iy2 <= GSL_MIN(iy + sy, met->ny - 1); iy2++) {
03309
03310                 /* Calculate Cartesian coordinates... */
03311                 double x1[3];
03312                 geo2cart(0.0, met->lon[ix3], met->lat[iy2], x1);
03313
03314                 /* Calculate weighting factor... */
03315                 float w = (float) exp(-DIST2(x0, x1) / tssq);
03316
03317                 /* Add data... */
03318                 wsum += w;
03319                 for (int ip = 0; ip < met->np; ip++) {
03320                     help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip];
03321                     help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip];
03322                     help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip];
03323                     help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip];
03324                 }
03325             }
03326         }
03327
03328         /* Normalize... */
03329         for (int ip = 0; ip < met->np; ip++) {
03330             help->t[ix][iy][ip] /= wsum;
03331             help->u[ix][iy][ip] /= wsum;
03332             help->v[ix][iy][ip] /= wsum;
03333             help->w[ix][iy][ip] /= wsum;
03334         }
03335     }
03336 }
03337
03338 /* Subtract background... */
03339 #pragma omp parallel for default(shared) collapse(3)
03340 for (int ix = 0; ix < met->nx; ix++)
03341     for (int iy = 0; iy < met->ny; iy++)
03342         for (int ip = 0; ip < met->np; ip++) {
03343             met->t[ix][iy][ip] -= help->t[ix][iy][ip];
03344             met->u[ix][iy][ip] -= help->u[ix][iy][ip];
03345             met->v[ix][iy][ip] -= help->v[ix][iy][ip];
03346             met->w[ix][iy][ip] -= help->w[ix][iy][ip];
03347         }
03348
03349 /* Free... */
03350 free(help);
03351 }

```


Here is the call graph for this function:



5.23.3.46 read_met_extrapolate() void read_met_extrapolate (
 met_t * met)

Extrapolate meteo data at lower boundary.

Definition at line 3355 of file libtrac.c.

```

03356     {
03357
03358     /* Set timer... */
03359     SELECT_TIMER("READ_MET_EXTRAPOLATE", "METPROC", NVTX_READ);
03360     LOG(2, "Extrapolate meteo data...");
03361
03362     /* Loop over columns... */
03363     #pragma omp parallel for default(shared) collapse(2)
03364     for (int ix = 0; ix < met->nx; ix++)
03365         for (int iy = 0; iy < met->ny; iy++) {
03366
03367         /* Find lowest valid data point... */
03368         int ip0;
03369         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
03370             if (!isfinite(met->t[ix][iy][ip0])
03371                 || !isfinite(met->u[ix][iy][ip0])
03372                 || !isfinite(met->v[ix][iy][ip0])
03373                 || !isfinite(met->w[ix][iy][ip0]))
03374                 break;
03375
03376         /* Extrapolate... */
03377         for (int ip = ip0; ip >= 0; ip--) {
03378             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
03379             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
03380             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
03381             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
03382             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
03383             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
03384             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
03385             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
03386         }
03387     }
03388 }
```

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

Calculate geopotential heights.

Definition at line 3392 of file libtrac.c.

```

03394     {
03395
03396     static float help[EP][EX][EY];
03397
```

```

03398 double logp[EP];
03399
03400 int dx = ctl->met_geopot_sx, dy = ctl->met_geopot_sy;
03401
03402 /* Set timer... */
03403 SELECT_TIMER("READ_MET_GEOPOT", "METPROC", NVTX_READ);
03404 LOG(2, "Calculate geopotential heights...");
03405
03406 /* Calculate log pressure... */
03407 #pragma omp parallel for default(shared)
03408 for (int ip = 0; ip < met->np; ip++)
03409     logp[ip] = log(met->p[ip]);
03410
03411 /* Apply hydrostatic equation to calculate geopotential heights... */
03412 #pragma omp parallel for default(shared) collapse(2)
03413 for (int ix = 0; ix < met->nx; ix++)
03414     for (int iy = 0; iy < met->ny; iy++) {
03415
03416         /* Get surface height and pressure... */
03417         double zs = met->zs[ix][iy];
03418         double lnps = log(met->ps[ix][iy]);
03419
03420         /* Get temperature and water vapor vmr at the surface... */
03421         int ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
03422         double ts = LIN(met->p[ip0], met->t[ix][iy][ip0], met->p[ip0 + 1],
03423             met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);
03424         double h2os = LIN(met->p[ip0], met->h2o[ix][iy][ip0], met->p[ip0 + 1],
03425             met->h2o[ix][iy][ip0 + 1], met->ps[ix][iy]);
03426
03427         /* Upper part of profile... */
03428         met->z[ix][iy][ip0 + 1]
03429             = (float) (zs +
03430                 ZDIFF(lnps, ts, h2os, logp[ip0 + 1],
03431                     met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]));
03432         for (int ip = ip0 + 2; ip < met->np; ip++)
03433             met->z[ix][iy][ip]
03434                 = (float) (met->z[ix][iy][ip - 1] +
03435                     ZDIFF(logp[ip - 1], met->t[ix][iy][ip - 1],
03436                         met->h2o[ix][iy][ip - 1], logp[ip],
03437                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03438
03439         /* Lower part of profile... */
03440         met->z[ix][iy][ip0]
03441             = (float) (zs +
03442                 ZDIFF(lnps, ts, h2os, logp[ip0],
03443                     met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]));
03444         for (int ip = ip0 - 1; ip >= 0; ip--)
03445             met->z[ix][iy][ip]
03446                 = (float) (met->z[ix][iy][ip + 1] +
03447                     ZDIFF(logp[ip + 1], met->t[ix][iy][ip + 1],
03448                         met->h2o[ix][iy][ip + 1], logp[ip],
03449                         met->t[ix][iy][ip], met->h2o[ix][iy][ip]));
03450     }
03451
03452 /* Check control parameters... */
03453 if (dx == 0 || dy == 0)
03454     return;
03455
03456 /* Default smoothing parameters... */
03457 if (dx < 0 || dy < 0) {
03458     if (fabs(met->lon[1] - met->lon[0]) < 0.5) {
03459         dx = 3;
03460         dy = 2;
03461     } else {
03462         dx = 6;
03463         dy = 4;
03464     }
03465 }
03466
03467 /* Calculate weights for smoothing... */
03468 float ws[dx + 1][dy + 1];
03469 #pragma omp parallel for default(shared) collapse(2)
03470 for (int ix = 0; ix <= dx; ix++)
03471     for (int iy = 0; iy < dy; iy++)
03472         ws[ix][iy] = (1.0f - (float) ix / (float) dx)
03473             * (1.0f - (float) iy / (float) dy);
03474
03475 /* Copy data... */
03476 #pragma omp parallel for default(shared) collapse(3)
03477 for (int ix = 0; ix < met->nx; ix++)
03478     for (int iy = 0; iy < met->ny; iy++)
03479         for (int ip = 0; ip < met->np; ip++)
03480             help[ip][ix][iy] = met->z[ix][iy][ip];
03481
03482 /* Horizontal smoothing... */
03483 #pragma omp parallel for default(shared) collapse(3)
03484 for (int ip = 0; ip < met->np; ip++)

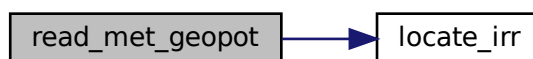
```

```

03485     for (int ix = 0; ix < met->nx; ix++)
03486     for (int iy = 0; iy < met->ny; iy++) {
03487         float res = 0, wsum = 0;
03488         int iy0 = GSL_MAX(iy - dy + 1, 0);
03489         int iy1 = GSL_MIN(iy + dy - 1, met->ny - 1);
03490         for (int ix2 = ix - dx + 1; ix2 <= ix + dx - 1; ++ix2) {
03491             int ix3 = ix2;
03492             if (ix3 < 0)
03493                 ix3 += met->nx;
03494             else if (ix3 >= met->nx)
03495                 ix3 -= met->nx;
03496             for (int iy2 = iy0; iy2 <= iy1; ++iy2)
03497                 if (isfinite(help[ip][ix3][iy2])) {
03498                     float w = ws[abs(ix - ix2)][abs(iy - iy2)];
03499                     res += w * help[ip][ix3][iy2];
03500                     wsum += w;
03501                 }
03502         }
03503         if (wsum > 0)
03504             met->z[ix][iy][ip] = res / wsum;
03505         else
03506             met->z[ix][iy][ip] = GSL_NAN;
03507     }
03508 }

```

Here is the call graph for this function:



5.23.3.48 read_met_grid() void read_met_grid (

```

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

```

Read coordinates of meteo data.

Definition at line 3512 of file libtrac.c.

```

03516     {
03517
03518         char levname[LEN], tstr[10];
03519
03520         double rtime, r2;
03521
03522         int varid, year2, mon2, day2, hour2, min2, sec2, year, mon, day, hour;
03523
03524         size_t np;
03525
03526         /* Set timer... */
03527         SELECT_TIMER("READ_MET_GRID", "INPUT", NVTX_READ);
03528         LOG(2, "Read meteo grid information...");
03529
03530         /* MPTRAC meteo files... */
03531         if (ctl->clams_met_data == 0) {
03532
03533             /* Get time from filename... */
03534             size_t len = strlen(filename);
03535             sprintf(tstr, "%.4s", &filename[len - 16]);
03536             year = atoi(tstr);
03537             sprintf(tstr, "%.2s", &filename[len - 11]);
03538             mon = atoi(tstr);
03539             sprintf(tstr, "%.2s", &filename[len - 8]);

```

```

03540     day = atoi(tstr);
03541     sprintf(tstr, "%.2s", &filename[len - 5]);
03542     hour = atoi(tstr);
03543     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03544
03545     /* Check time information from data file... */
03546     if (nc_inq_varid(ncid, "time", &varid) == NC_NOERR) {
03547         NC(nc_get_var_double(ncid, varid, &rttime));
03548         if (fabs(year * 10000. + mon * 100. + day + hour / 24. - rttime) > 1.0)
03549             WARN("Time information in meteo file does not match filename!");
03550     } else
03551         WARN("Time information in meteo file is missing!");
03552 }
03553
03554 /* CLaMS meteo files... */
03555 else {
03556
03557     /* Read time from file... */
03558     NC_GET_DOUBLE("time", &rttime, 0);
03559
03560     /* Get time from filename (considering the century)... */
03561     if (rttime < 0)
03562         sprintf(tstr, "19%.2s", &filename[strlen(filename) - 11]);
03563     else
03564         sprintf(tstr, "20%.2s", &filename[strlen(filename) - 11]);
03565     year = atoi(tstr);
03566     sprintf(tstr, "%.2s", &filename[strlen(filename) - 9]);
03567     mon = atoi(tstr);
03568     sprintf(tstr, "%.2s", &filename[strlen(filename) - 7]);
03569     day = atoi(tstr);
03570     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
03571     hour = atoi(tstr);
03572     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
03573 }
03574
03575 /* Check time... */
03576 if (year < 1900 || year > 2100 || mon < 1 || mon > 12
03577     || day < 1 || day > 31 || hour < 0 || hour > 23)
03578     ERRMSG("Cannot read time from filename!");
03579 jsec2time(met->time, &year2, &mon2, &day2, &hour2, &min2, &sec2, &r2);
03580 LOG(2, "Time: %.2f (%d-%02d-%02d, %02d:%02d UTC)",
03581     met->time, year2, mon2, day2, hour2, min2);
03582
03583 /* Get grid dimensions... */
03584 NC_INQ_DIM("lon", &met->nx, 2, EX);
03585 LOG(2, "Number of longitudes: %d", met->nx);
03586
03587 NC_INQ_DIM("lat", &met->ny, 2, EY);
03588 LOG(2, "Number of latitudes: %d", met->ny);
03589
03590 if (ctl->vert_coord_meteo == 0) {
03591     int dimid;
03592     sprintf(levname, "lev");
03593     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR)
03594         sprintf(levname, "plev");
03595 } else
03596     sprintf(levname, "hybrid");
03597 NC_INQ_DIM(levname, &met->np, 1, EP);
03598 if (met->np == 1) {
03599     int dimid;
03600     sprintf(levname, "lev_2");
03601     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
03602         sprintf(levname, "plev");
03603         nc_inq_dimid(ncid, levname, &dimid);
03604     }
03605     NC(nc_inq_dimlen(ncid, dimid, &np));
03606     met->np = (int) np;
03607 }
03608 LOG(2, "Number of levels: %d", met->np);
03609 if (met->np < 2 || met->np > EP)
03610     ERRMSG("Number of levels out of range!");
03611
03612 /* Read longitudes and latitudes... */
03613 NC_GET_DOUBLE("lon", met->lon, 1);
03614 LOG(2, "Longitudes: %g, %g ... %g deg",
03615     met->lon[0], met->lon[1], met->lon[met->nx - 1]);
03616 NC_GET_DOUBLE("lat", met->lat, 1);
03617 LOG(2, "Latitudes: %g, %g ... %g deg",
03618     met->lat[0], met->lat[1], met->lat[met->ny - 1]);
03619
03620 /* Read pressure levels... */
03621 if (ctl->met_np <= 0) {
03622     NC_GET_DOUBLE(levname, met->p, 1);
03623     for (int ip = 0; ip < met->np; ip++)
03624         met->p[ip] /= 100.;
03625     LOG(2, "Altitude levels: %g, %g ... %g km",
03626         Z(met->p[0]), Z(met->p[1]), Z(met->p[met->np - 1]));

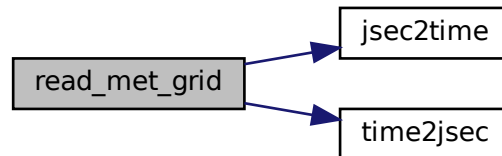
```

```

03627     LOG(2, "Pressure levels: %g, %g ... %g hPa",
03628         met->p[0], met->p[1], met->p[met->np - 1]);
03629 }
03630 }

```

Here is the call graph for this function:



5.23.3.49 read_met_levels() void read_met_levels (

```

    int ncid,
    ctl_t * ctl,
    met_t * met )

```

Read meteo data on vertical levels.

Definition at line 3634 of file libtrac.c.

```

03637     {
03638
03639     /* Set timer... */
03640     SELECT_TIMER("READ_MET_LEVELS", "INPUT", NVTX_READ);
03641     LOG(2, "Read level data...");
03642
03643     /* MPTRAC meteo data... */
03644     if (ctl->clams_met_data == 0) {
03645
03646         /* Read meteo data... */
03647         if (!read_met_nc_3d(ncid, "t", "T", ctl, met, met->t, 1.0, 1))
03648             ERRMSG("Cannot read temperature!");
03649         if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03650             ERRMSG("Cannot read zonal wind!");
03651         if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03652             ERRMSG("Cannot read meridional wind!");
03653         if (!read_met_nc_3d(ncid, "w", "W", ctl, met, met->w, 0.01f, 1))
03654             WARN("Cannot read vertical velocity!");
03655         if (!read_met_nc_3d
03656             (ncid, "q", "Q", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03657             WARN("Cannot read specific humidity!");
03658         if (!read_met_nc_3d
03659             (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03660             WARN("Cannot read ozone data!");
03661         if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03662             if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03663                 WARN("Cannot read cloud liquid water content!");
03664             if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03665                 WARN("Cannot read cloud ice water content!");
03666         }
03667         if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03668             if (!read_met_nc_3d
03669                 (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03670                  ctl->met_cloud == 2))
03671                 WARN("Cannot read cloud rain water content!");
03672             if (!read_met_nc_3d
03673                 (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03674                  ctl->met_cloud == 2))
03675                 WARN("Cannot read cloud snow water content!");
03676         }
03677     }
03678 }

```

```

03677     if (ctl->met_relhum) {
03678         if (!read_met_nc_3d(ncid, "rh", "RH", ctl, met, met->h2o, 0.01f, 1))
03679             WARN("Cannot read relative humidity!");
03680 #pragma omp parallel for default(shared) collapse(2)
03681         for (int ix = 0; ix < met->nx; ix++)
03682             for (int iy = 0; iy < met->ny; iy++)
03683                 for (int ip = 0; ip < met->np; ip++) {
03684                     double pw = met->h2o[ix][iy][ip] * PSAT(met->t[ix][iy][ip]);
03685                     met->h2o[ix][iy][ip] =
03686                         (float) (pw / (met->p[ip] - (1.0 - EPS) * pw));
03687                 }
03688     }
03689
03690     /* Transfer from model levels to pressure levels... */
03691     if (ctl->met_np > 0) {
03692
03693         /* Read pressure on model levels... */
03694         if (!read_met_nc_3d(ncid, "pl", "PL", ctl, met, met->p1, 0.01f, 1))
03695             ERRMSG("Cannot read pressure on model levels!");
03696
03697         /* Vertical interpolation from model to pressure levels... */
03698         read_met_ml2pl(ctl, met, met->t);
03699         read_met_ml2pl(ctl, met, met->u);
03700         read_met_ml2pl(ctl, met, met->v);
03701         read_met_ml2pl(ctl, met, met->w);
03702         read_met_ml2pl(ctl, met, met->h2o);
03703         read_met_ml2pl(ctl, met, met->o3);
03704         read_met_ml2pl(ctl, met, met->lwc);
03705         read_met_ml2pl(ctl, met, met->iwc);
03706
03707         /* Set new pressure levels... */
03708         met->np = ctl->met_np;
03709         for (int ip = 0; ip < met->np; ip++)
03710             met->p[ip] = ctl->met_p[ip];
03711     }
03712 }
03713
03714 /* CLaMS meteo data... */
03715 else if (ctl->clams_met_data == 1) {
03716
03717     /* Read meteorological data... */
03718     if (!read_met_nc_3d(ncid, "t", "TEMP", ctl, met, met->t, 1.0, 1))
03719         ERRMSG("Cannot read temperature!");
03720     if (!read_met_nc_3d(ncid, "u", "U", ctl, met, met->u, 1.0, 1))
03721         ERRMSG("Cannot read zonal wind!");
03722     if (!read_met_nc_3d(ncid, "v", "V", ctl, met, met->v, 1.0, 1))
03723         ERRMSG("Cannot read meridional wind!");
03724     if (!read_met_nc_3d(ncid, "w", "OMEGA", ctl, met, met->w, 0.01f, 1))
03725         WARN("Cannot read vertical velocity!");
03726     if (!read_met_nc_3d(ncid, "zeta", "ZETA", ctl, met, met->zeta, 1.0, 1))
03727         WARN("Cannot read ZETA in meteo data!");
03728     if (ctl->vert_vel == 1) {
03729         if (!read_met_nc_3d
03730             (ncid, "ZETA_DOT_TOT", "zeta_dot_clr", ctl, met, met->zeta_dot,
03731              0.00001157407f, 1)) {
03732             if (!read_met_nc_3d
03733                 (ncid, "ZETA_DOT_TOT", "ZETA_DOT_clr", ctl, met, met->zeta_dot,
03734                  0.00001157407f, 1)) {
03735                 WARN("Cannot read vertical velocity!");
03736             }
03737         }
03738     }
03739
03740     if (!read_met_nc_3d
03741         (ncid, "sh", "SH", ctl, met, met->h2o, (float) (MA / MH2O), 1))
03742         WARN("Cannot read specific humidity!");
03743     if (!read_met_nc_3d
03744         (ncid, "o3", "O3", ctl, met, met->o3, (float) (MA / MO3), 1))
03745         WARN("Cannot read ozone data!");
03746     if (ctl->met_cloud == 1 || ctl->met_cloud == 3) {
03747         if (!read_met_nc_3d(ncid, "clwc", "CLWC", ctl, met, met->lwc, 1.0, 1))
03748             WARN("Cannot read cloud liquid water content!");
03749         if (!read_met_nc_3d(ncid, "ciwc", "CIWC", ctl, met, met->iwc, 1.0, 1))
03750             WARN("Cannot read cloud ice water content!");
03751     }
03752     if (ctl->met_cloud == 2 || ctl->met_cloud == 3) {
03753         if (!read_met_nc_3d
03754             (ncid, "crwc", "CRWC", ctl, met, met->lwc, 1.0,
03755              ctl->met_cloud == 2))
03756             WARN("Cannot read cloud rain water content!");
03757         if (!read_met_nc_3d
03758             (ncid, "cswc", "CSWC", ctl, met, met->iwc, 1.0,
03759              ctl->met_cloud == 2))
03760             WARN("Cannot read cloud snow water content!");
03761     }
03762
03763     /* Transfer from model levels to pressure levels... */

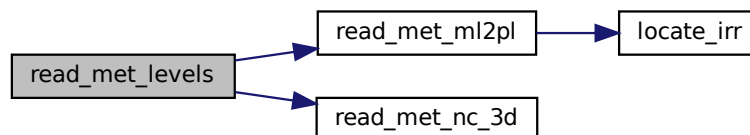
```

```

03764     if (ctl->met_np > 0) {
03765
03766         /* Read pressure on model levels... */
03767         if (!read_met_nc_3d(ncid, "p1", "PRESS", ctl, met, met->p1, 1.0, 1))
03768             ERRMSG("Cannot read pressure on model levels!");
03769
03770         /* Vertical interpolation from model to pressure levels... */
03771         read_met_ml2pl(ctl, met, met->t);
03772         read_met_ml2pl(ctl, met, met->u);
03773         read_met_ml2pl(ctl, met, met->v);
03774         read_met_ml2pl(ctl, met, met->w);
03775         read_met_ml2pl(ctl, met, met->h2o);
03776         read_met_ml2pl(ctl, met, met->o3);
03777         read_met_ml2pl(ctl, met, met->lwc);
03778         read_met_ml2pl(ctl, met, met->iwc);
03779         if (ctl->vert_vel == 1) {
03780             read_met_ml2pl(ctl, met, met->zeta);
03781             read_met_ml2pl(ctl, met, met->zeta_dot);
03782         }
03783
03784         /* Set new pressure levels... */
03785         met->np = ctl->met_np;
03786         for (int ip = 0; ip < met->np; ip++)
03787             met->p[ip] = ctl->met_p[ip];
03788
03789         /* Create a pressure field... */
03790         for (int i = 0; i < met->nx; i++)
03791             for (int j = 0; j < met->ny; j++)
03792                 for (int k = 0; k < met->np; k++) {
03793                     met->patp[i][j][k] = (float) met->p[k];
03794                 }
03795     }
03796 } else
03797     ERRMSG("Meteo data format unknown!");
03798
03799 /* Check ordering of pressure levels... */
03800 for (int ip = 1; ip < met->np; ip++)
03801     if (met->p[ip - 1] < met->p[ip])
03802         ERRMSG("Pressure levels must be descending!");
03803 }

```

Here is the call graph for this function:



5.23.3.50 read_met_ml2pl() void read_met_ml2pl (

```

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

```

Convert meteo data from model levels to pressure levels.

Definition at line 3807 of file `libtrac.c`.

```

03810     {
03811
03812         double aux[EP], p[EP];
03813
03814         /* Set timer... */
03815         SELECT_TIMER("READ_MET_ML2PL", "METPROC", NVTX_READ);
03816         LOG(2, "Interpolate meteo data to pressure levels...");

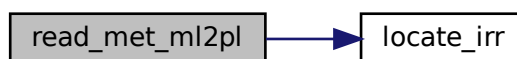
```

```

03817
03818  /* Loop over columns... */
03819 #pragma omp parallel for default(shared) private(aux,p) collapse(2)
03820 for (int ix = 0; ix < met->nx; ix++)
03821   for (int iy = 0; iy < met->ny; iy++) {
03822
03823     /* Copy pressure profile... */
03824     for (int ip = 0; ip < met->np; ip++)
03825       p[ip] = met->pl[ix][iy][ip];
03826
03827     /* Interpolate... */
03828     for (int ip = 0; ip < ctl->met_np; ip++) {
03829       double pt = ctl->met_p[ip];
03830       if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
03831         pt = p[0];
03832       else if ((pt > p[met->np - 1] && p[1] > p[0])
03833               || (pt < p[met->np - 1] && p[1] < p[0]))
03834         pt = p[met->np - 1];
03835       int ip2 = locate_irr(p, met->np, pt);
03836       aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
03837                   p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
03838     }
03839
03840     /* Copy data... */
03841     for (int ip = 0; ip < ctl->met_np; ip++)
03842       var[ix][iy][ip] = (float) aux[ip];
03843   }
03844 }

```

Here is the call graph for this function:



5.23.3.51 read_met_nc_2d() int read_met_nc_2d (

```

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

```

Read and convert 2D variable from meteo data file.

Definition at line 3848 of file libtrac.c.

```

03856     {
03857
03858     char varsel[LEN];
03859
03860     float offset, scalfac;
03861
03862     int varid;
03863
03864     /* Check if variable exists... */
03865     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03866       if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03867         WARN("Cannot read 2-D variable: %s or %s", varname, varname2);
03868         return 0;
03869       } else {

```



```

03870     sprintf(varsel, "%s", varname2);
03871 } else
03872     sprintf(varsel, "%s", varname);
03873
03874 /* Read packed data... */
03875 if (ctl->met_nc_scale
03876     && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03877     && nc_get_att_float(ncid, varid, "scale_factor",
03878                         &scalfac) == NC_NOERR) {
03879
03880     /* Allocate... */
03881     short *help;
03882     ALLOC(help, short,
03883           EX * EY * EP);
03884
03885     /* Read fill value and missing value... */
03886     short fillval, missval;
03887     if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03888         fillval = 0;
03889     if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
03890         missval = 0;
03891
03892     /* Write info... */
03893     LOG(2, "Read 2-D variable: %s"
03894         " (FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
03895         varsel, fillval, missval, scalfac, offset);
03896
03897     /* Read data... */
03898     NC(nc_get_var_short(ncid, varid, help));
03899
03900     /* Copy and check data... */
03901 #pragma omp parallel for default(shared) num_threads(12)
03902     for (int ix = 0; ix < met->nx; ix++)
03903         for (int iy = 0; iy < met->ny; iy++) {
03904             if (init)
03905                 dest[ix][iy] = 0;
03906             short aux = help[ARRAY_2D(iy, ix, met->nx)];
03907             if ((fillval == 0 || aux != fillval)
03908                 && (missval == 0 || aux != missval)
03909                 && fabsf(aux * scalfac + offset) < 1e14f)
03910                 dest[ix][iy] += scl * (aux * scalfac + offset);
03911             else
03912                 dest[ix][iy] = GSL_NAN;
03913         }
03914
03915     /* Free... */
03916     free(help);
03917 }
03918
03919 /* Unpacked data... */
03920 else {
03921
03922     /* Allocate... */
03923     float *help;
03924     ALLOC(help, float,
03925           EX * EY);
03926
03927     /* Read fill value and missing value... */
03928     float fillval, missval;
03929     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
03930         fillval = 0;
03931     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
03932         missval = 0;
03933
03934     /* Write info... */
03935     LOG(2, "Read 2-D variable: %s (FILL = %g, MISS = %g)",
03936         varsel, fillval, missval);
03937
03938     /* Read data... */
03939     NC(nc_get_var_float(ncid, varid, help));
03940
03941     /* Copy and check data... */
03942 #pragma omp parallel for default(shared) num_threads(12)
03943     for (int ix = 0; ix < met->nx; ix++)
03944         for (int iy = 0; iy < met->ny; iy++) {
03945             if (init)
03946                 dest[ix][iy] = 0;
03947             float aux = help[ARRAY_2D(iy, ix, met->nx)];
03948             if ((fillval == 0 || aux != fillval)
03949                 && (missval == 0 || aux != missval)
03950                 && fabsf(aux) < 1e14f)
03951                 dest[ix][iy] += scl * aux;
03952             else
03953                 dest[ix][iy] = GSL_NAN;
03954         }
03955
03956     /* Free... */

```

```

03957     free(help);
03958 }
03959
03960 /* Return... */
03961 return 1;
03962 }

```

5.23.3.52 read_met_nc_3d() int read_met_nc_3d (

```

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

```

Read and convert 3D variable from meteo data file.

Definition at line 3966 of file libtrac.c.

```

03974     {
03975
03976     char varsel[LEN];
03977
03978     float offset, scalfac;
03979
03980     int varid;
03981
03982     /* Check if variable exists... */
03983     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
03984         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR) {
03985             WARN("Cannot read 3-D variable: %s or %s", varname, varname2);
03986             return 0;
03987         } else {
03988             sprintf(varsel, "%s", varname2);
03989         } else
03990             sprintf(varsel, "%s", varname);
03991
03992     /* Read packed data... */
03993     if (ctl->met_nc_scale
03994         && nc_get_att_float(ncid, varid, "add_offset", &offset) == NC_NOERR
03995         && nc_get_att_float(ncid, varid, "scale_factor",
03996                             &scalfac) == NC_NOERR) {
03997
03998         /* Allocate... */
03999         short *help;
04000         ALLOC(help, short,
04001              EX * EY * EP);
04002
04003         /* Read fill value and missing value... */
04004         short fillval, missval;
04005         if (nc_get_att_short(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
04006             fillval = 0;
04007         if (nc_get_att_short(ncid, varid, "missing_value", &missval) != NC_NOERR)
04008             missval = 0;
04009
04010         /* Write info... */
04011         LOG(2, "Read 3-D variable: %s "
04012             "(FILL = %d, MISS = %d, SCALE = %g, OFFSET = %g)",
04013            varsel, fillval, missval, scalfac, offset);
04014
04015         /* Read data... */
04016         NC(nc_get_var_short(ncid, varid, help));
04017
04018         /* Copy and check data... */
04019 #pragma omp parallel for default(shared) num_threads(12)
04020         for (int ix = 0; ix < met->nx; ix++)
04021             for (int iy = 0; iy < met->ny; iy++)
04022                 for (int ip = 0; ip < met->np; ip++) {
04023                     if (init)
04024                         dest[ix][iy][ip] = 0;
04025                     short aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04026                     if ((fillval == 0 || aux != fillval)
04027                         && (missval == 0 || aux != missval)
04028                         && fabsf(aux * scalfac + offset) < 1e14f)

```

```

04029         dest[ix][iy][ip] += scl * (aux * scalfac + offset);
04030     else
04031         dest[ix][iy][ip] = GSL_NAN;
04032     }
04033
04034     /* Free... */
04035     free(help);
04036 }
04037
04038 /* Unpacked data... */
04039 else {
04040
04041     /* Allocate... */
04042     float *help;
04043     ALLOC(help, float,
04044           EX * EY * EP);
04045
04046     /* Read fill value and missing value... */
04047     float fillval, missval;
04048     if (nc_get_att_float(ncid, varid, "_FillValue", &fillval) != NC_NOERR)
04049         fillval = 0;
04050     if (nc_get_att_float(ncid, varid, "missing_value", &missval) != NC_NOERR)
04051         missval = 0;
04052
04053     /* Write info... */
04054     LOG(2, "Read 3-D variable: %s (FILL = %g, MISS = %g)",
04055         varsel, fillval, missval);
04056
04057     /* Read data... */
04058     NC(nc_get_var_float(ncid, varid, help));
04059
04060     /* Copy and check data... */
04061     #pragma omp parallel for default(shared) num_threads(12)
04062     for (int ix = 0; ix < met->nx; ix++)
04063         for (int iy = 0; iy < met->ny; iy++)
04064             for (int ip = 0; ip < met->np; ip++) {
04065                 if (init)
04066                     dest[ix][iy][ip] = 0;
04067                 float aux = help[ARRAY_3D(ip, iy, met->ny, ix, met->nx)];
04068                 if ((fillval == 0 || aux != fillval)
04069                     && (missval == 0 || aux != missval)
04070                     && fabsf(aux) < 1e14f)
04071                     dest[ix][iy][ip] += scl * aux;
04072                 else
04073                     dest[ix][iy][ip] = GSL_NAN;
04074             }
04075
04076     /* Free... */
04077     free(help);
04078 }
04079
04080 /* Return... */
04081 return 1;
04082 }

```

5.23.3.53 read_met_pbl() void read_met_pbl (
 met_t * met)

Calculate pressure of the boundary layer.

Definition at line 4086 of file libtrac.c.

```

04087     {
04088
04089     /* Set timer... */
04090     SELECT_TIMER("READ_MET_PBL", "METPROC", NVTX_READ);
04091     LOG(2, "Calculate planetary boundary layer...");
04092
04093     /* Parameters used to estimate the height of the PBL
04094        (e.g., Vogelesang and Holtslag, 1996; Seidel et al., 2012)... */
04095     const double rib_crit = 0.25, dz = 0.05, umin = 5.0;
04096
04097     /* Loop over grid points... */
04098     #pragma omp parallel for default(shared) collapse(2)
04099     for (int ix = 0; ix < met->nx; ix++)
04100         for (int iy = 0; iy < met->ny; iy++) {
04101
04102         /* Set bottom level of PBL... */
04103         double pbl_bot = met->ps[ix][iy] + DZ2DP(dz, met->ps[ix][iy]);
04104

```

```

04105      /* Find lowest level near the bottom... */
04106      int ip;
04107      for (ip = 1; ip < met->np; ip++)
04108          if (met->p[ip] < pbl_bot)
04109              break;
04110
04111      /* Get near surface data... */
04112      double zs = LIN(met->p[ip - 1], met->z[ix][iy][ip - 1],
04113                     met->p[ip], met->z[ix][iy][ip], pbl_bot);
04114      double ts = LIN(met->p[ip - 1], met->t[ix][iy][ip - 1],
04115                     met->p[ip], met->t[ix][iy][ip], pbl_bot);
04116      double us = LIN(met->p[ip - 1], met->u[ix][iy][ip - 1],
04117                     met->p[ip], met->u[ix][iy][ip], pbl_bot);
04118      double vs = LIN(met->p[ip - 1], met->v[ix][iy][ip - 1],
04119                     met->p[ip], met->v[ix][iy][ip], pbl_bot);
04120      double h2os = LIN(met->p[ip - 1], met->h2o[ix][iy][ip - 1],
04121                       met->p[ip], met->h2o[ix][iy][ip], pbl_bot);
04122      double tvs = THETA_VIRT(pbl_bot, ts, h2os);
04123
04124      /* Init... */
04125      double rib_old = 0;
04126
04127      /* Loop over levels... */
04128      for (; ip < met->np; ip++) {
04129
04130          /* Get squared horizontal wind speed... */
04131          double vh2
04132              = SQR(met->u[ix][iy][ip] - us) + SQR(met->v[ix][iy][ip] - vs);
04133          vh2 = GSL_MAX(vh2, SQR(umin));
04134
04135          /* Calculate bulk Richardson number... */
04136          double rib = GO * 1e3 * (met->z[ix][iy][ip] - zs) / tvs
04137              * (THETA_VIRT(met->p[ip], met->t[ix][iy][ip],
04138                           met->h2o[ix][iy][ip]) - tvs) / vh2;
04139
04140          /* Check for critical value... */
04141          if (rib >= rib_crit) {
04142              met->pbl[ix][iy] = (float) (LIN(rib_old, met->p[ip - 1],
04143                                             rib, met->p[ip], rib_crit));
04144              if (met->pbl[ix][iy] > pbl_bot)
04145                  met->pbl[ix][iy] = (float) pbl_bot;
04146              break;
04147          }
04148
04149          /* Save Richardson number... */
04150          rib_old = rib;
04151      }
04152  }
04153 }

```

5.23.3.54 read_met_periodic() void read_met_periodic (
 met_t * met)

Create meteo data with periodic boundary conditions.

Definition at line 4157 of file libtrac.c.

```

04158      {
04159
04160          /* Set timer... */
04161          SELECT_TIMER("READ_MET_PERIODIC", "METPROC", NVTX_READ);
04162          LOG(2, "Apply periodic boundary conditions...");
04163
04164          /* Check longitudes... */
04165          if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
04166                  + met->lon[1] - met->lon[0] - 360) < 0.01))
04167              return;
04168
04169          /* Increase longitude counter... */
04170          if ((++met->nx) > EX)
04171              ERRMSG("Cannot create periodic boundary conditions!");
04172
04173          /* Set longitude... */
04174          met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];
04175
04176          /* Loop over latitudes and pressure levels... */
04177          #pragma omp parallel for default(shared)
04178          for (int iy = 0; iy < met->ny; iy++) {
04179              met->ps[met->nx - 1][iy] = met->ps[0][iy];
04180              met->zs[met->nx - 1][iy] = met->zs[0][iy];

```

```

04181     met->ts[met->nx - 1][iy] = met->ts[0][iy];
04182     met->us[met->nx - 1][iy] = met->us[0][iy];
04183     met->vs[met->nx - 1][iy] = met->vs[0][iy];
04184     for (int ip = 0; ip < met->np; ip++) {
04185         met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
04186         met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
04187         met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
04188         met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
04189         met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
04190         met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
04191         met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
04192         met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
04193     }
04194 }
04195 }

```

5.23.3.55 read_met_pv() void read_met_pv (
 met_t * met)

Calculate potential vorticity.

Definition at line 4199 of file libtrac.c.

```

04200     {
04201
04202     double pows[EP];
04203
04204     /* Set timer... */
04205     SELECT_TIMER("READ_MET_PV", "METPROC", NVTX_READ);
04206     LOG(2, "Calculate potential vorticity...");
04207
04208     /* Set powers... */
04209     #pragma omp parallel for default(shared)
04210     for (int ip = 0; ip < met->np; ip++)
04211         pows[ip] = pow(1000. / met->p[ip], 0.286);
04212
04213     /* Loop over grid points... */
04214     #pragma omp parallel for default(shared)
04215     for (int ix = 0; ix < met->nx; ix++) {
04216
04217         /* Set indices... */
04218         int ix0 = GSL_MAX(ix - 1, 0);
04219         int ix1 = GSL_MIN(ix + 1, met->nx - 1);
04220
04221         /* Loop over grid points... */
04222         for (int iy = 0; iy < met->ny; iy++) {
04223
04224             /* Set indices... */
04225             int iy0 = GSL_MAX(iy - 1, 0);
04226             int iy1 = GSL_MIN(iy + 1, met->ny - 1);
04227
04228             /* Set auxiliary variables... */
04229             double latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
04230             double dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
04231             double dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
04232             double c0 = cos(met->lat[iy0] / 180. * M_PI);
04233             double c1 = cos(met->lat[iy1] / 180. * M_PI);
04234             double cr = cos(latr / 180. * M_PI);
04235             double vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
04236
04237             /* Loop over grid points... */
04238             for (int ip = 0; ip < met->np; ip++) {
04239
04240                 /* Get gradients in longitude... */
04241                 double dtdx
04242                     = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
04243                 double dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
04244
04245                 /* Get gradients in latitude... */
04246                 double dtdy
04247                     = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
04248                 double dudx
04249                     = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
04250
04251                 /* Set indices... */
04252                 int ip0 = GSL_MAX(ip - 1, 0);
04253                 int ip1 = GSL_MIN(ip + 1, met->np - 1);
04254
04255                 /* Get gradients in pressure... */
04256                 double dtdp, dudp, dvdp;

```

```

04257     double dp0 = 100. * (met->p[ip] - met->p[ip0]);
04258     double dp1 = 100. * (met->p[ip1] - met->p[ip]);
04259     if (ip != ip0 && ip != ip1) {
04260         double denom = dp0 * dp1 * (dp0 + dp1);
04261         dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
04262             - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
04263             + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
04264             / denom;
04265         dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
04266             - dp1 * dp1 * met->u[ix][iy][ip0]
04267             + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
04268             / denom;
04269         dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
04270             - dp1 * dp1 * met->v[ix][iy][ip0]
04271             + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
04272             / denom;
04273     } else {
04274         double denom = dp0 + dp1;
04275         dtdp =
04276             (met->t[ix][iy][ip1] * pows[ip1] -
04277             met->t[ix][iy][ip0] * pows[ip0]) / denom;
04278         dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
04279         dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
04280     }
04281
04282     /* Calculate PV... */
04283     met->pv[ix][iy][ip] = (float)
04284         (1e6 * G0 *
04285         (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
04286 }
04287 }
04288 }
04289
04290 /* Fix for polar regions... */
04291 #pragma omp parallel for default(shared)
04292 for (int ix = 0; ix < met->nx; ix++)
04293     for (int ip = 0; ip < met->np; ip++) {
04294         met->pv[ix][0][ip]
04295             = met->pv[ix][1][ip]
04296             = met->pv[ix][2][ip];
04297         met->pv[ix][met->ny - 1][ip]
04298             = met->pv[ix][met->ny - 2][ip]
04299             = met->pv[ix][met->ny - 3][ip];
04300     }
04301 }

```

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

Downsampling of meteo data.

Definition at line 4305 of file libtrac.c.

```

04307     {
04308
04309     met_t *help;
04310
04311     /* Check parameters... */
04312     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
04313         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
04314         return;
04315
04316     /* Set timer... */
04317     SELECT_TIMER("READ_MET_SAMPLE", "METPROC", NVTX_READ);
04318     LOG(2, "Downsampling of meteo data...");
04319
04320     /* Allocate... */
04321     ALLOC(help, met_t, 1);
04322
04323     /* Copy data... */
04324     help->nx = met->nx;
04325     help->ny = met->ny;
04326     help->np = met->np;
04327     memcpy(help->lon, met->lon, sizeof(met->lon));
04328     memcpy(help->lat, met->lat, sizeof(met->lat));
04329     memcpy(help->p, met->p, sizeof(met->p));
04330
04331     /* Smoothing... */

```

```

04332     for (int ix = 0; ix < met->nx; ix += ctl->met_dx) {
04333         for (int iy = 0; iy < met->ny; iy += ctl->met_dy) {
04334             for (int ip = 0; ip < met->np; ip += ctl->met_dp) {
04335                 help->ps[ix][iy] = 0;
04336                 help->zs[ix][iy] = 0;
04337                 help->ts[ix][iy] = 0;
04338                 help->us[ix][iy] = 0;
04339                 help->vs[ix][iy] = 0;
04340                 help->t[ix][iy][ip] = 0;
04341                 help->u[ix][iy][ip] = 0;
04342                 help->v[ix][iy][ip] = 0;
04343                 help->w[ix][iy][ip] = 0;
04344                 help->h2o[ix][iy][ip] = 0;
04345                 help->o3[ix][iy][ip] = 0;
04346                 help->lwc[ix][iy][ip] = 0;
04347                 help->iwc[ix][iy][ip] = 0;
04348                 float wsum = 0;
04349                 for (int ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1;
04350                     ix2++) {
04351                     int ix3 = ix2;
04352                     if (ix3 < 0)
04353                         ix3 += met->nx;
04354                     else if (ix3 >= met->nx)
04355                         ix3 -= met->nx;
04356
04357                     for (int iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
04358                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
04359                         for (int ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
04360                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
04361                             float w = (1.0f - (float) abs(ix - ix2) / (float) ctl->met_sx)
04362                                 * (1.0f - (float) abs(iy - iy2) / (float) ctl->met_sy)
04363                                 * (1.0f - (float) abs(ip - ip2) / (float) ctl->met_sp);
04364                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
04365                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
04366                             help->ts[ix][iy] += w * met->ts[ix3][iy2];
04367                             help->us[ix][iy] += w * met->us[ix3][iy2];
04368                             help->vs[ix][iy] += w * met->vs[ix3][iy2];
04369                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
04370                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
04371                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
04372                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
04373                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
04374                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
04375                             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
04376                             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
04377                             wsum += w;
04378                         }
04379                     }
04380                 help->ps[ix][iy] /= wsum;
04381                 help->zs[ix][iy] /= wsum;
04382                 help->ts[ix][iy] /= wsum;
04383                 help->us[ix][iy] /= wsum;
04384                 help->vs[ix][iy] /= wsum;
04385                 help->t[ix][iy][ip] /= wsum;
04386                 help->u[ix][iy][ip] /= wsum;
04387                 help->v[ix][iy][ip] /= wsum;
04388                 help->w[ix][iy][ip] /= wsum;
04389                 help->h2o[ix][iy][ip] /= wsum;
04390                 help->o3[ix][iy][ip] /= wsum;
04391                 help->lwc[ix][iy][ip] /= wsum;
04392                 help->iwc[ix][iy][ip] /= wsum;
04393             }
04394         }
04395     }
04396
04397     /* Downsampling... */
04398     met->nx = 0;
04399     for (int ix = 0; ix < help->nx; ix += ctl->met_dx) {
04400         met->lon[met->nx] = help->lon[ix];
04401         met->ny = 0;
04402         for (int iy = 0; iy < help->ny; iy += ctl->met_dy) {
04403             met->lat[met->ny] = help->lat[iy];
04404             met->ps[met->nx][met->ny] = help->ps[ix][iy];
04405             met->zs[met->nx][met->ny] = help->zs[ix][iy];
04406             met->ts[met->nx][met->ny] = help->ts[ix][iy];
04407             met->us[met->nx][met->ny] = help->us[ix][iy];
04408             met->vs[met->nx][met->ny] = help->vs[ix][iy];
04409             met->np = 0;
04410             for (int ip = 0; ip < help->np; ip += ctl->met_dp) {
04411                 met->p[met->np] = help->p[ip];
04412                 met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
04413                 met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
04414                 met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
04415                 met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
04416                 met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
04417                 met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
04418                 met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];

```

```

04419         met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
04420         met->np++;
04421     }
04422     met->ny++;
04423 }
04424     met->nx++;
04425 }
04426
04427 /* Free... */
04428 free(help);
04429 }

```

5.23.3.57 read_met_surface() void read_met_surface (

```

    int ncid,
    met_t * met,
    ctl_t * ctl )

```

Read surface data.

Definition at line 4433 of file libtrac.c.

```

04436     {
04437
04438     /* Set timer... */
04439     SELECT_TIMER("READ_MET_SURFACE", "INPUT", NVTX_READ);
04440     LOG(2, "Read surface data...");
04441
04442     /* MPTRAC meteo data... */
04443     if (ctl->clams_meteo_data == 0) {
04444
04445         /* Read surface pressure... */
04446         if (!read_met_nc_2d(ncid, "lnsp", "LNSP", ctl, met, met->ps, 1.0f, 1)) {
04447             if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04448                 if (!read_met_nc_2d(ncid, "sp", "SP", ctl, met, met->ps, 0.01f, 1)) {
04449                     WARN("Cannot not read surface pressure data (use lowest level)!");
04450                     for (int ix = 0; ix < met->nx; ix++)
04451                         for (int iy = 0; iy < met->ny; iy++)
04452                             met->ps[ix][iy] = (float) met->p[0];
04453                 }
04454             }
04455         } else
04456             for (int ix = 0; ix < met->nx; ix++)
04457                 for (int iy = 0; iy < met->ny; iy++)
04458                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
04459
04460         /* Read geopotential height at the surface... */
04461         if (!read_met_nc_2d
04462             (ncid, "z", "Z", ctl, met, met->zs, (float) (1. / (1000. * G0)), 1))
04463             if (!read_met_nc_2d
04464                 (ncid, "zm", "ZM", ctl, met, met->zs, (float) (1. / 1000.), 1))
04465                 WARN("Cannot read surface geopotential height!");
04466
04467         /* Read temperature at the surface... */
04468         if (!read_met_nc_2d(ncid, "t2m", "T2M", ctl, met, met->ts, 1.0, 1))
04469             WARN("Cannot read surface temperature!");
04470
04471         /* Read zonal wind at the surface... */
04472         if (!read_met_nc_2d(ncid, "u10m", "U10M", ctl, met, met->us, 1.0, 1))
04473             WARN("Cannot read surface zonal wind!");
04474
04475         /* Read meridional wind at the surface... */
04476         if (!read_met_nc_2d(ncid, "v10m", "V10M", ctl, met, met->vs, 1.0, 1))
04477             WARN("Cannot read surface meridional wind!");
04478     }
04479
04480     /* CLaMS meteo data... */
04481     else {
04482
04483         /* Read surface pressure... */
04484         if (!read_met_nc_2d(ncid, "ps", "PS", ctl, met, met->ps, 0.01f, 1)) {
04485             WARN("Cannot not read surface pressure data (use lowest level)!");
04486             for (int ix = 0; ix < met->nx; ix++)
04487                 for (int iy = 0; iy < met->ny; iy++)
04488                     met->ps[ix][iy] = (float) met->p[0];
04489         }
04490
04491         /* Read geopotential height at the surface
04492            (use lowermost level of 3-D data field)... */
04493         float *help;

```

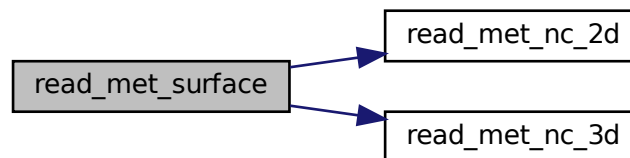


```

04494     ALLOC(help, float,
04495           EX * EY * EP);
04496     memcpy(help, met->p1, sizeof(met->p1));
04497     if (!read_met_nc_3d
04498         (ncid, "gph", "GPH", ctl, met, met->p1, (float) (1e-3 / G0), 1)) {
04499         ERRMSG("Cannot read geopotential height!");
04500     } else
04501         for (int ix = 0; ix < met->nx; ix++)
04502             for (int iy = 0; iy < met->ny; iy++)
04503                 met->zs[ix][iy] = met->p1[ix][iy][0];
04504     memcpy(met->p1, help, sizeof(met->p1));
04505     free(help);
04506
04507     /* Read temperature at the surface... */
04508     if (!read_met_nc_2d(ncid, "t2", "T2", ctl, met, met->ts, 1.0, 1))
04509         WARN("Cannot read surface temperature!");
04510
04511     /* Read zonal wind at the surface... */
04512     if (!read_met_nc_2d(ncid, "u10", "U10", ctl, met, met->us, 1.0, 1))
04513         WARN("Cannot read surface zonal wind!");
04514
04515     /* Read meridional wind at the surface... */
04516     if (!read_met_nc_2d(ncid, "v10", "V10", ctl, met, met->vs, 1.0, 1))
04517         WARN("Cannot read surface meridional wind!");
04518 }
04519 }

```

Here is the call graph for this function:



5.23.3.58 read_met_tropo() void read_met_tropo (

```

    ctl_t * ctl,
    clim_t * clim,
    met_t * met )

```

Calculate tropopause data.

Definition at line 4523 of file libtrac.c.

```

04526     {
04527
04528     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
04529           th2[200], z[EP], z2[200];
04530
04531     /* Set timer... */
04532     SELECT_TIMER("READ_MET_TROPO", "METPROC", NVTX_READ);
04533     LOG(2, "Calculate tropopause...");
04534
04535     /* Get altitude and pressure profiles... */
04536     #pragma omp parallel for default(shared)
04537     for (int iz = 0; iz < met->np; iz++)
04538         z[iz] = Z(met->p[iz]);
04539     #pragma omp parallel for default(shared)
04540     for (int iz = 0; iz <= 190; iz++) {
04541         z2[iz] = 4.5 + 0.1 * iz;
04542         p2[iz] = P(z2[iz]);
04543     }

```

```

04544
04545  /* Do not calculate tropopause... */
04546  if (ctl->met_tropo == 0)
04547 #pragma omp parallel for default(shared) collapse(2)
04548    for (int ix = 0; ix < met->nx; ix++)
04549      for (int iy = 0; iy < met->ny; iy++)
04550        met->pt[ix][iy] = GSL_NAN;
04551
04552  /* Use tropopause climatology... */
04553  else if (ctl->met_tropo == 1) {
04554 #pragma omp parallel for default(shared) collapse(2)
04555    for (int ix = 0; ix < met->nx; ix++)
04556      for (int iy = 0; iy < met->ny; iy++)
04557        met->pt[ix][iy] = (float) clim_tropo(clim, met->time, met->lat[iy]);
04558  }
04559
04560  /* Use cold point... */
04561  else if (ctl->met_tropo == 2) {
04562
04563    /* Loop over grid points... */
04564 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04565    for (int ix = 0; ix < met->nx; ix++)
04566      for (int iy = 0; iy < met->ny; iy++) {
04567
04568        /* Interpolate temperature profile... */
04569        for (int iz = 0; iz < met->np; iz++)
04570          t[iz] = met->t[ix][iy][iz];
04571        spline(z, t, met->np, z2, t2, 171, ctl->met_tropo_spline);
04572
04573        /* Find minimum... */
04574        int iz = (int) gsl_stats_min_index(t2, 1, 171);
04575        if (iz > 0 && iz < 170)
04576          met->pt[ix][iy] = (float) p2[iz];
04577        else
04578          met->pt[ix][iy] = GSL_NAN;
04579      }
04580  }
04581
04582  /* Use WMO definition... */
04583  else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
04584
04585    /* Loop over grid points... */
04586 #pragma omp parallel for default(shared) private(t,t2) collapse(2)
04587    for (int ix = 0; ix < met->nx; ix++)
04588      for (int iy = 0; iy < met->ny; iy++) {
04589
04590        /* Interpolate temperature profile... */
04591        int iz;
04592        for (iz = 0; iz < met->np; iz++)
04593          t[iz] = met->t[ix][iy][iz];
04594        spline(z, t, met->np, z2, t2, 191, ctl->met_tropo_spline);
04595
04596        /* Find 1st tropopause... */
04597        met->pt[ix][iy] = GSL_NAN;
04598        for (iz = 0; iz <= 170; iz++) {
04599          int found = 1;
04600          for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04601            if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >
04602                ctl->met_tropo_lapse) {
04603              found = 0;
04604              break;
04605            }
04606          if (found) {
04607            if (iz > 0 && iz < 170)
04608              met->pt[ix][iy] = (float) p2[iz];
04609            break;
04610          }
04611        }
04612
04613        /* Find 2nd tropopause... */
04614        if (ctl->met_tropo == 4) {
04615          met->pt[ix][iy] = GSL_NAN;
04616          for (; iz <= 170; iz++) {
04617            int found = 1;
04618            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev_sep; iz2++)
04619              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) <
04620                  ctl->met_tropo_lapse_sep) {
04621                found = 0;
04622                break;
04623              }
04624            if (found)
04625              break;
04626          }
04627          for (; iz <= 170; iz++) {
04628            int found = 1;
04629            for (int iz2 = iz + 1; iz2 <= iz + ctl->met_tropo_nlev; iz2++)
04630              if (LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]) >

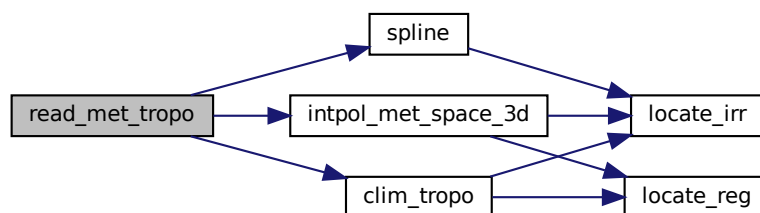
```

```

04631         ctl->met_tropo_lapse) {
04632             found = 0;
04633             break;
04634         }
04635         if (found) {
04636             if (iz > 0 && iz < 170)
04637                 met->pt[ix][iy] = (float) p2[iz];
04638             break;
04639         }
04640     }
04641 }
04642 }
04643 }
04644
04645 /* Use dynamical tropopause... */
04646 else if (ctl->met_tropo == 5) {
04647     /* Loop over grid points... */
04648     #pragma omp parallel for default(shared) private(pv,pv2,th,th2) collapse(2)
04649     for (int ix = 0; ix < met->nx; ix++)
04650         for (int iy = 0; iy < met->ny; iy++) {
04651             /* Interpolate potential vorticity profile... */
04652             for (int iz = 0; iz < met->np; iz++)
04653                 pv[iz] = met->pv[ix][iy][iz];
04654             spline(z, pv, met->np, z2, pv2, 171, ctl->met_tropo_spline);
04655
04656             /* Interpolate potential temperature profile... */
04657             for (int iz = 0; iz < met->np; iz++)
04658                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
04659             spline(z, th, met->np, z2, th2, 171, ctl->met_tropo_spline);
04660
04661             /* Find dynamical tropopause... */
04662             met->pt[ix][iy] = GSL_NAN;
04663             for (int iz = 0; iz <= 170; iz++)
04664                 if (fabs(pv2[iz]) >= ctl->met_tropo_pv
04665                     || th2[iz] >= ctl->met_tropo_theta) {
04666                     if (iz > 0 && iz < 170)
04667                         met->pt[ix][iy] = (float) p2[iz];
04668                     break;
04669                 }
04670         }
04671     }
04672 }
04673 }
04674
04675 else
04676     ERRMSG("Cannot calculate tropopause!");
04677
04678 /* Interpolate temperature, geopotential height, and water vapor vmr... */
04679 #pragma omp parallel for default(shared) collapse(2)
04680 for (int ix = 0; ix < met->nx; ix++)
04681     for (int iy = 0; iy < met->ny; iy++) {
04682         double h2ot, tt, zt;
04683         INTPOL_INIT;
04684         intpol_met_space_3d(met, met->t, met->pt[ix][iy], met->lon[ix],
04685                             met->lat[iy], &tt, ci, cw, 1);
04686         intpol_met_space_3d(met, met->z, met->pt[ix][iy], met->lon[ix],
04687                             met->lat[iy], &z, ci, cw, 0);
04688         intpol_met_space_3d(met, met->h2o, met->pt[ix][iy], met->lon[ix],
04689                             met->lat[iy], &h2ot, ci, cw, 0);
04690         met->tt[ix][iy] = (float) tt;
04691         met->z[ix][iy] = (float) z;
04692         met->h2ot[ix][iy] = (float) h2ot;
04693     }
04694 }

```

Here is the call graph for this function:



```

5.23.3.59 read_obs() void read_obs (
    char * filename,
    double * rt,
    double * rz,
    double * rlon,
    double * rlat,
    double * robs,
    int * nobs )

```

Read observation data.

Definition at line 4698 of file libtrac.c.

```

04705     {
04706
04707     FILE *in;
04708
04709     char line[LEN];
04710
04711     /* Open observation data file... */
04712     LOG(1, "Read observation data: %s", filename);
04713     if (!(in = fopen(filename, "r")))
04714         ERRMSG("Cannot open file!");
04715
04716     /* Read observations... */
04717     while (fgets(line, LEN, in))
04718         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt[*nobs], &rz[*nobs],
04719                 &rlon[*nobs], &rlat[*nobs], &robs[*nobs]) == 5)
04720             if ((++(*nobs)) >= NOBS)
04721                 ERRMSG("Too many observations!");
04722
04723     /* Close observation data file... */
04724     fclose(in);
04725
04726     /* Check time... */
04727     for (int i = 1; i < *nobs; i++)
04728         if (rt[i] < rt[i - 1])
04729             ERRMSG("Time must be ascending!");
04730
04731     /* Write info... */
04732     int n = *nobs;
04733     double mini, maxi;
04734     LOG(2, "Number of observations: %d", *nobs);
04735     gsl_stats_minmax(&mini, &maxi, rt, 1, (size_t) n);
04736     LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
04737     gsl_stats_minmax(&mini, &maxi, rz, 1, (size_t) n);
04738     LOG(2, "Altitude range: %g ... %g km", mini, maxi);
04739     gsl_stats_minmax(&mini, &maxi, rlon, 1, (size_t) n);
04740     LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
04741     gsl_stats_minmax(&mini, &maxi, rlat, 1, (size_t) n);
04742     LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
04743     gsl_stats_minmax(&mini, &maxi, robs, 1, (size_t) n);
04744     LOG(2, "Observation range: %g ... %g", mini, maxi);
04745 }

```

```

5.23.3.60 scan_ctl() double scan_ctl (
    const char * filename,
    int argc,
    char * argv[],
    const char * varname,
    int arridx,
    const char * defvalue,
    char * value )

```

Read a control parameter from file or command line.

Definition at line 4749 of file libtrac.c.

```

04756         {
04757
04758     FILE *in = NULL;
04759
04760     char fullname1[LEN], fullname2[LEN], rval[LEN];
04761
04762     int contain = 0, i;
04763
04764     /* Open file... */
04765     if (filename[strlen(filename) - 1] != '-')
04766         if (!(in = fopen(filename, "r")))
04767             ERRMSG("Cannot open file!");
04768
04769     /* Set full variable name... */
04770     if (arridx >= 0) {
04771         sprintf(fullname1, "%s[%d]", varname, arridx);
04772         sprintf(fullname2, "%s[*]", varname);
04773     } else {
04774         sprintf(fullname1, "%s", varname);
04775         sprintf(fullname2, "%s", varname);
04776     }
04777
04778     /* Read data... */
04779     if (in != NULL) {
04780         char dummy[LEN], line[LEN], rvarname[LEN];
04781         while (fgets(line, LEN, in)) {
04782             if (sscanf(line, "%4999s %4999s", rvarname, dummy, rval) == 3)
04783                 if (strcasecmp(rvarname, fullname1) == 0 ||
04784                     strcasecmp(rvarname, fullname2) == 0) {
04785                     contain = 1;
04786                     break;
04787                 }
04788         }
04789     }
04790     for (i = 1; i < argc - 1; i++)
04791         if (strcasecmp(argv[i], fullname1) == 0 ||
04792             strcasecmp(argv[i], fullname2) == 0) {
04793             sprintf(rval, "%s", argv[i + 1]);
04794             contain = 1;
04795             break;
04796         }
04797
04798     /* Close file... */
04799     if (in != NULL)
04800         fclose(in);
04801
04802     /* Check for missing variables... */
04803     if (!contain) {
04804         if (strlen(defvalue) > 0)
04805             sprintf(rval, "%s", defvalue);
04806         else
04807             ERRMSG("Missing variable %s!\n", fullname1);
04808     }
04809
04810     /* Write info... */
04811     LOG(1, "%s = %s", fullname1, rval);
04812
04813     /* Return values... */
04814     if (value != NULL)
04815         sprintf(value, "%s", rval);
04816     return atof(rval);
04817 }

```

5.23.3.61 sedi() double sedi (
 double p,
 double T,
 double rp,
 double rhop)

Calculate sedimentation velocity.

Definition at line 4821 of file libtrac.c.

```

04825     {
04826
04827     /* Convert particle radius from microns to m... */
04828     rp *= 1e-6;
04829

```

```

04830  /* Density of dry air [kg / m^3]... */
04831  double rho = RHO(p, T);
04832
04833  /* Dynamic viscosity of air [kg / (m s)]... */
04834  double eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
04835
04836  /* Thermal velocity of an air molecule [m / s]... */
04837  double v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
04838
04839  /* Mean free path of an air molecule [m]... */
04840  double lambda = 2. * eta / (rho * v);
04841
04842  /* Knudsen number for air (dimensionless)... */
04843  double K = lambda / rp;
04844
04845  /* Cunningham slip-flow correction (dimensionless)... */
04846  double G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
04847
04848  /* Sedimentation velocity [m / s]... */
04849  return 2. * SQR(rp) * (rhop - rho) * G0 / (9. * eta) * G;
04850 }

```

5.23.3.62 spline() void spline (

```

    double * x,
    double * y,
    int n,
    double * x2,
    double * y2,
    int n2,
    int method )

```

Spline interpolation.

Definition at line 4854 of file libtrac.c.

```

04861  {
04862
04863  /* Cubic spline interpolation... */
04864  if (method == 1) {
04865
04866      /* Allocate... */
04867      gsl_interp_accel *acc;
04868      gsl_spline *s;
04869      acc = gsl_interp_accel_alloc();
04870      s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
04871
04872      /* Interpolate profile... */
04873      gsl_spline_init(s, x, y, (size_t) n);
04874      for (int i = 0; i < n2; i++)
04875          if (x2[i] <= x[0])
04876              y2[i] = y[0];
04877          else if (x2[i] >= x[n - 1])
04878              y2[i] = y[n - 1];
04879          else
04880              y2[i] = gsl_spline_eval(s, x2[i], acc);
04881
04882      /* Free... */
04883      gsl_spline_free(s);
04884      gsl_interp_accel_free(acc);
04885  }
04886
04887  /* Linear interpolation... */
04888  else {
04889      for (int i = 0; i < n2; i++)
04890          if (x2[i] <= x[0])
04891              y2[i] = y[0];
04892          else if (x2[i] >= x[n - 1])
04893              y2[i] = y[n - 1];
04894          else {
04895              int idx = locate_irr(x, n, x2[i]);
04896              y2[i] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], x2[i]);
04897          }
04898  }
04899  }

```

Here is the call graph for this function:



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

Calculate standard deviation.

Definition at line 4903 of file libtrac.c.

```

04905     {
04906
04907     if (n <= 0)
04908         return 0;
04909
04910     float mean = 0, var = 0;
04911
04912     for (int i = 0; i < n; ++i) {
04913         mean += data[i];
04914         var += SQR(data[i]);
04915     }
04916
04917     var = var / (float) n - SQR(mean / (float) n);
04918
04919     return (var > 0 ? sqrtf(var) : 0);
04920 }
  
```

5.23.3.64 sza() double sza (
 double sec,
 double lon,
 double lat)

Calculate solar zenith angle.

Definition at line 4924 of file libtrac.c.

```

04927     {
04928
04929     double D, dec, e, g, GMST, h, L, LST, q, ra;
04930
04931     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
04932     D = sec / 86400 - 0.5;
04933
04934     /* Geocentric apparent ecliptic longitude [rad]... */
04935     g = (357.529 + 0.98560028 * D) * M_PI / 180;
04936     q = 280.459 + 0.98564736 * D;
04937     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
04938
04939     /* Mean obliquity of the ecliptic [rad]... */
04940     e = (23.439 - 0.00000036 * D) * M_PI / 180;
04941
04942     /* Declination [rad]... */
04943     dec = asin(sin(e) * sin(L));
04944 }
  
```

```

04945  /* Right ascension [rad]... */
04946  ra = atan2(cos(e) * sin(L), cos(L));
04947
04948  /* Greenwich Mean Sidereal Time [h]... */
04949  GMST = 18.697374558 + 24.06570982441908 * D;
04950
04951  /* Local Sidereal Time [h]... */
04952  LST = GMST + lon / 15;
04953
04954  /* Hour angle [rad]... */
04955  h = LST / 12 * M_PI - ra;
04956
04957  /* Convert latitude... */
04958  lat *= M_PI / 180;
04959
04960  /* Return solar zenith angle [rad]... */
04961  return acos(sin(lat) * sin(dec) + cos(lat) * cos(dec) * cos(h));
04962 }

```

5.23.3.65 time2jsec() void time2jsec (

```

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

```

Convert date to seconds.

Definition at line 4966 of file libtrac.c.

```

04974      {
04975
04976      struct tm t0, t1;
04977
04978      t0.tm_year = 100;
04979      t0.tm_mon = 0;
04980      t0.tm_mday = 1;
04981      t0.tm_hour = 0;
04982      t0.tm_min = 0;
04983      t0.tm_sec = 0;
04984
04985      t1.tm_year = year - 1900;
04986      t1.tm_mon = mon - 1;
04987      t1.tm_mday = day;
04988      t1.tm_hour = hour;
04989      t1.tm_min = min;
04990      t1.tm_sec = sec;
04991
04992      *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04993 }

```

5.23.3.66 timer() void timer (

```

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

```

Measure wall-clock time.

Definition at line 4997 of file libtrac.c.

```

05000      {
05001
05002      static char names[NTIMER][100], groups[NTIMER][100];
05003
05004      static double rt_name[NTIMER], rt_group[NTIMER],

```



```

05005     rt_min[NTIMER], rt_max[NTIMER], dt, t0, t1;
05006
05007     static int iname = -1, igrp = -1, nname, ngroup, ct_name[NTIMER];
05008
05009     /* Get time... */
05010     t1 = omp_get_wtime();
05011     dt = t1 - t0;
05012
05013     /* Add elapsed time to current timers... */
05014     if (iname >= 0) {
05015         rt_name[iname] += dt;
05016         rt_min[iname] = (ct_name[iname] <= 0 ? dt : GSL_MIN(rt_min[iname], dt));
05017         rt_max[iname] = (ct_name[iname] <= 0 ? dt : GSL_MAX(rt_max[iname], dt));
05018         ct_name[iname]++;
05019     }
05020     if (igrp >= 0)
05021         rt_group[igrp] += t1 - t0;
05022
05023     /* Report timers... */
05024     if (output) {
05025         for (int i = 0; i < nname; i++)
05026             LOG(1, "TIMER_%s = %.3f s (min= %g s, mean= %g s, "
05027                 " max= %g s, n= %d)", names[i], rt_name[i], rt_min[i],
05028                 rt_name[i] / ct_name[i], rt_max[i], ct_name[i]);
05029         for (int i = 0; i < ngroup; i++)
05030             LOG(1, "TIMER_GROUP_%s = %.3f s", groups[i], rt_group[i]);
05031         double total = 0.0;
05032         for (int i = 0; i < nname; i++)
05033             total += rt_name[i];
05034         LOG(1, "TIMER_TOTAL = %.3f s", total);
05035     }
05036
05037     /* Identify IDs of next timer... */
05038     for (iname = 0; iname < nname; iname++)
05039         if (strcmp(name, names[iname]) == 0)
05040             break;
05041     for (igrp = 0; igrp < ngroup; igrp++)
05042         if (strcmp(group, groups[igrp]) == 0)
05043             break;
05044
05045     /* Check whether this is a new timer... */
05046     if (iname >= nname) {
05047         sprintf(names[iname], "%s", name);
05048         if ((++nname) > NTIMER)
05049             ERRMSG("Too many timers!");
05050     }
05051
05052     /* Check whether this is a new group... */
05053     if (igrp >= ngroup) {
05054         sprintf(groups[igrp], "%s", group);
05055         if ((++ngroup) > NTIMER)
05056             ERRMSG("Too many groups!");
05057     }
05058
05059     /* Save starting time... */
05060     t0 = t1;
05061 }

```

5.23.3.67 tropo_weight() double tropo_weight (
 clim_t * clim,
 double t,
 double lat,
 double p)

Get weighting factor based on tropopause distance.

Definition at line 5065 of file libtrac.c.

```

05069     {
05070
05071     /* Get tropopause pressure... */
05072     double pt = clim_tropo(clim, t, lat);
05073
05074     /* Get pressure range... */
05075     double p1 = pt * 0.866877899;
05076     double p0 = pt / 0.866877899;
05077
05078     /* Get weighting factor... */

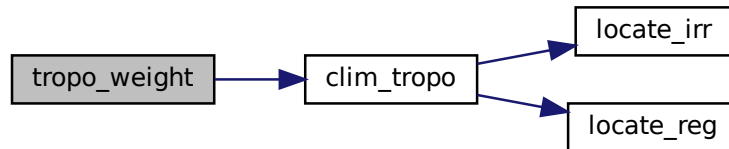
```

```

05079  if (p > p0)
05080      return 1;
05081  else if (p < p1)
05082      return 0;
05083  else
05084      return LIN(p0, 1.0, p1, 0.0, p);
05085  }

```

Here is the call graph for this function:



5.23.3.68 write_atm() void write_atm (

const char * filename,

ctl_t * ctl,

atm_t * atm,

double t)

Write atmospheric data.

Definition at line 5089 of file libtrac.c.

```

05093  {
05094
05095  /* Set timer... */
05096  SELECT_TIMER("WRITE_ATM", "OUTPUT", NVTX_WRITE);
05097
05098  /* Write info... */
05099  LOG(1, "Write atmospheric data: %s", filename);
05100
05101  /* Write ASCII data... */
05102  if (ctl->atm_type == 0)
05103      write_atm_asc(filename, ctl, atm, t);
05104
05105  /* Write binary data... */
05106  else if (ctl->atm_type == 1)
05107      write_atm_bin(filename, ctl, atm);
05108
05109  /* Write netCDF data... */
05110  else if (ctl->atm_type == 2)
05111      write_atm_nc(filename, ctl, atm);
05112
05113  /* Write CLaMS data... */
05114  else if (ctl->atm_type == 3)
05115      write_atm_clams(ctl, atm, t);
05116
05117  /* Error... */
05118  else
05119      ERRMSG("Atmospheric data type not supported!");
05120
05121  /* Write info... */
05122  double mini, maxi;
05123  LOG(2, "Number of particles: %d", atm->np);
05124  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
05125  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
05126  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
05127  LOG(2, "Altitude range: %g ... %g km", Z(maxi), Z(mini));
05128  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
05129  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);

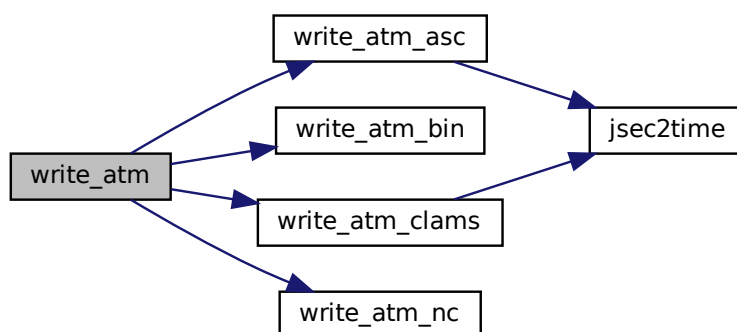
```

```

05130 LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
05131 gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
05132 LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
05133 for (int iq = 0; iq < ctl->nq; iq++) {
05134     char msg[LEN];
05135     sprintf(msg, "Quantity %s range: %s ... %s %s",
05136             ctl->qnt_name[iq], ctl->qnt_format[iq],
05137             ctl->qnt_format[iq], ctl->qnt_unit[iq]);
05138     gsl_stats_minmax(&mini, &maxi, atm->q[iq], 1, (size_t) atm->np);
05139     LOG(2, msg, mini, maxi);
05140 }
05141 }

```

Here is the call graph for this function:



5.23.3.69 write_atm_asc() void write_atm_asc (

```

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

```

Write atmospheric data in ASCII format.

Definition at line 5145 of file libtrac.c.

```

05149 {
05150
05151     FILE *out;
05152
05153     /* Set time interval for output... */
05154     double t0 = t - 0.5 * ctl->dt_mod;
05155     double t1 = t + 0.5 * ctl->dt_mod;
05156
05157     /* Check if gnuplot output is requested... */
05158     if (ctl->atm_gpfile[0] != '-') {
05159
05160         /* Create gnuplot pipe... */
05161         if (!(out = popen("gnuplot", "w")))
05162             ERRMSG("Cannot create pipe to gnuplot!");
05163
05164         /* Set plot filename... */
05165         fprintf(out, "set out \"%s.png\"\n", filename);
05166
05167         /* Set time string... */
05168         double r;
05169         int year, mon, day, hour, min, sec;
05170         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05171         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
05172                 year, mon, day, hour, min);

```

```

05173
05174     /* Dump gnuplot file to pipe... */
05175     FILE *in;
05176     if (!(in = fopen(ctl->atm_gpfile, "r")))
05177         ERRMSG("Cannot open file!");
05178     char line[LEN];
05179     while (fgets(line, LEN, in))
05180         fprintf(out, "%s", line);
05181     fclose(in);
05182 }
05183
05184 else {
05185     /* Create file... */
05186     if (!(out = fopen(filename, "w")))
05187         ERRMSG("Cannot create file!");
05188 }
05189
05190
05191 /* Write header... */
05192 fprintf(out,
05193         "# $1 = time [s]\n"
05194         "# $2 = altitude [km]\n"
05195         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05196 for (int iq = 0; iq < ctl->nq; iq++)
05197     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
05198             ctl->qnt_unit[iq]);
05199 fprintf(out, "\n");
05200
05201 /* Write data... */
05202 for (int ip = 0; ip < atm->np; ip += ctl->atm_stride) {
05203
05204     /* Check time... */
05205     if (ctl->atm_filter == 2 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05206         continue;
05207
05208     /* Write output... */
05209     fprintf(out, "%.2f %g %g %g", atm->time[ip], 2(atm->p[ip]),
05210             atm->lon[ip], atm->lat[ip]);
05211     for (int iq = 0; iq < ctl->nq; iq++) {
05212         fprintf(out, " ");
05213         if (ctl->atm_filter == 1 && (atm->time[ip] < t0 || atm->time[ip] > t1))
05214             fprintf(out, ctl->qnt_format[iq], GSL_NAN);
05215         else
05216             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
05217     }
05218     fprintf(out, "\n");
05219 }
05220
05221 /* Close file... */
05222 fclose(out);
05223 }

```

Here is the call graph for this function:



5.23.3.70 write_atm_bin() void write_atm_bin (
 const char * filename,
 ctl_t * ctl,
 atm_t * atm)

Write atmospheric data in binary format.

Definition at line 5227 of file libtrac.c.

```

05230     {
05231
05232     FILE *out;
05233
05234     /* Create file... */
05235     if (!(out = fopen(filename, "w")))
05236         ERRMSG("Cannot create file!");
05237
05238     /* Write version of binary data... */
05239     int version = 100;
05240     FWRITE(&version, int,
05241           1,
05242           out);
05243
05244     /* Write data... */
05245     FWRITE(&atm->np, int,
05246           1,
05247           out);
05248     FWRITE(atm->time, double,
05249           (size_t) atm->np,
05250           out);
05251     FWRITE(atm->p, double,
05252           (size_t) atm->np,
05253           out);
05254     FWRITE(atm->lon, double,
05255           (size_t) atm->np,
05256           out);
05257     FWRITE(atm->lat, double,
05258           (size_t) atm->np,
05259           out);
05260     for (int iq = 0; iq < ctl->nq; iq++)
05261         FWRITE(atm->q[iq], double,
05262               (size_t) atm->np,
05263               out);
05264
05265     /* Write final flag... */
05266     int final = 999;
05267     FWRITE(&final, int,
05268           1,
05269           out);
05270
05271     /* Close file... */
05272     fclose(out);
05273 }

```

5.23.3.71 write_atm_clams() void write_atm_clams (
 ctl_t * ctl,
 atm_t * atm,
 double t)

Write atmospheric data in CLaMS format.

Definition at line 5277 of file libtrac.c.

```

05280     {
05281
05282     /* Global Counter... */
05283     static size_t out_cnt = 0;
05284
05285     char filename_out[2 * LEN] = "./traj_fix_3d_YYYYMMDDHH_YYYYMMDDHH.nc";
05286
05287     double r, r_start, r_stop;
05288
05289     int year, mon, day, hour, min, sec;
05290     int year_start, mon_start, day_start, hour_start, min_start, sec_start;
05291     int year_stop, mon_stop, day_stop, hour_stop, min_stop, sec_stop;
05292     int ncid, varid, tid, pid, cid, zid, dim_ids[2];
05293
05294     /* time, nparc */
05295     size_t start[2], count[2];
05296
05297     /* Determine start and stop times of calculation... */
05298     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
05299     jsec2time(ctl->t_start, &year_start, &mon_start, &day_start, &hour_start,
05300             &min_start, &sec_start, &r_start);
05301     jsec2time(ctl->t_stop, &year_stop, &mon_stop, &day_stop, &hour_stop,
05302             &min_stop, &sec_stop, &r_stop);
05303 }

```

```

05304  /* Set filename... */
05305  sprintf(filename_out,
05306          "./traj_fix_3d_%02d%02d%02d%02d_%02d%02d%02d%02d.nc",
05307          year_start % 100, mon_start, day_start, hour_start,
05308          year_stop % 100, mon_stop, day_stop, hour_stop);
05309  printf("Write traj file: %s\n", filename_out);
05310
05311  /* Define hyperslap for the traj_file... */
05312  start[0] = out_cnt;
05313  start[1] = 0;
05314  count[0] = 1;
05315  count[1] = (size_t) atm->np;
05316
05317  /* Create the file at the first timestep... */
05318  if (out_cnt == 0) {
05319
05320      /* Create file... */
05321      nc_create(filename_out, NC_CLOBBER, &ncid);
05322
05323      /* Define dimensions... */
05324      NC(nc_def_dim(ncid, "time", NC_UNLIMITED, &tid));
05325      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05326      NC(nc_def_dim(ncid, "TMDT", 7, &cid));
05327      dim_ids[0] = tid;
05328      dim_ids[1] = pid;
05329
05330      /* Define variables and their attributes... */
05331      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05332               "seconds since 2000-01-01 00:00:00 UTC");
05333      NC_DEF_VAR("LAT", NC_DOUBLE, 2, dim_ids, "Latitude", "deg");
05334      NC_DEF_VAR("LON", NC_DOUBLE, 2, dim_ids, "Longitude", "deg");
05335      NC_DEF_VAR("PRESS", NC_DOUBLE, 2, dim_ids, "Pressure", "hPa");
05336      NC_DEF_VAR("ZETA", NC_DOUBLE, 2, dim_ids, "Zeta", "K");
05337      for (int iq = 0; iq < ctl->nq; iq++)
05338          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05339                   ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05340
05341      /* Define global attributes... */
05342      NC_PUT_ATT_GLOBAL("exp_VERTCOORD_name", "zeta");
05343      NC_PUT_ATT_GLOBAL("model", "MPTRAC");
05344
05345      /* End definitions... */
05346      NC(nc_enddef(ncid));
05347      NC(nc_close(ncid));
05348  }
05349
05350  /* Increment global counter to change hyperslap... */
05351  out_cnt++;
05352
05353  /* Open file... */
05354  NC(nc_open(filename_out, NC_WRITE, &ncid));
05355
05356  /* Write data... */
05357  NC_PUT_DOUBLE("time", atm->time, 1);
05358  NC_PUT_DOUBLE("LAT", atm->lat, 1);
05359  NC_PUT_DOUBLE("LON", atm->lon, 1);
05360  NC_PUT_DOUBLE("PRESS", atm->p, 1);
05361  if (ctl->vert_coord_ap == 1) {
05362      NC_PUT_DOUBLE("ZETA", atm->zeta, 1);
05363  } else if (ctl->qnt_zeta >= 0) {
05364      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 1);
05365  }
05366  for (int iq = 0; iq < ctl->nq; iq++)
05367      NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 1);
05368
05369  /* Close file... */
05370  NC(nc_close(ncid));
05371
05372  /* At the last time step create the init_fix_YYYYMMDDHH file... */
05373  if ((year == year_stop) && (mon == mon_stop)
05374      && (day == day_stop) && (hour == hour_stop)) {
05375
05376      /* Set filename... */
05377      char filename_init[2 * LEN] = "./init_fix_YYYYMMDDHH.nc";
05378      sprintf(filename_init, "./init_fix_%02d%02d%02d%02d.nc",
05379              year_stop % 100, mon_stop, day_stop, hour_stop);
05380      printf("Write init file: %s\n", filename_init);
05381
05382      /* Create file... */
05383      nc_create(filename_init, NC_CLOBBER, &ncid);
05384
05385      /* Define dimensions... */
05386      NC(nc_def_dim(ncid, "time", 1, &tid));
05387      NC(nc_def_dim(ncid, "NPARTS", (size_t) atm->np, &pid));
05388      dim_ids[0] = tid;
05389      dim_ids[1] = pid;
05390

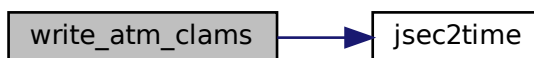
```

```

05391      /* Define variables and their attributes... */
05392      NC_DEF_VAR("time", NC_DOUBLE, 1, &tid, "Time",
05393                "seconds since 2000-01-01 00:00:00 UTC");
05394      NC_DEF_VAR("LAT", NC_DOUBLE, 1, &pid, "Latitude", "deg");
05395      NC_DEF_VAR("LON", NC_DOUBLE, 1, &pid, "Longitude", "deg");
05396      NC_DEF_VAR("PRESS", NC_DOUBLE, 1, &pid, "Pressure", "hPa");
05397      NC_DEF_VAR("ZETA", NC_DOUBLE, 1, &pid, "Zeta", "K");
05398      NC_DEF_VAR("ZETA_GRID", NC_DOUBLE, 1, &zid, "levels", "K");
05399      NC_DEF_VAR("ZETA_DELTA", NC_DOUBLE, 1, &zid, "Width of zeta levels", "K");
05400      for (int iq = 0; iq < ctl->nq; iq++)
05401          NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 2, dim_ids,
05402                    ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05403
05404      /* Define global attributes... */
05405      NC_PUT_ATT_GLOBAL("exp_VERTCOOR_name", "zeta");
05406      NC_PUT_ATT_GLOBAL("model", "MPTRAC");
05407
05408      /* End definitions... */
05409      NC(nc_enddef(ncid));
05410
05411      /* Write data... */
05412      NC_PUT_DOUBLE("time", atm->time, 0);
05413      NC_PUT_DOUBLE("LAT", atm->lat, 0);
05414      NC_PUT_DOUBLE("LON", atm->lon, 0);
05415      NC_PUT_DOUBLE("PRESS", atm->p, 0);
05416      NC_PUT_DOUBLE("ZETA", atm->q[ctl->qnt_zeta], 0);
05417      for (int iq = 0; iq < ctl->nq; iq++)
05418          NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05419
05420      /* Close file... */
05421      NC(nc_close(ncid));
05422  }
05423 }

```

Here is the call graph for this function:



5.23.3.72 write_atm_nc() void write_atm_nc (

 const char * filename,

 ctl_t * ctl,

 atm_t * atm)

Write atmospheric data in netCDF format.

Definition at line 5427 of file libtrac.c.

```

05430      {
05431
05432      int ncid, obsid, varid;
05433
05434      size_t start[2], count[2];
05435
05436      /* Create file... */
05437      nc_create(filename, NC_CLOBBER, &ncid);
05438
05439      /* Define dimensions... */
05440      NC(nc_def_dim(ncid, "obs", (size_t) atm->np, &obsid));
05441
05442      /* Define variables and their attributes... */
05443      NC_DEF_VAR("time", NC_DOUBLE, 1, &obsid, "time",
05444                "seconds since 2000-01-01 00:00:00 UTC");
05445      NC_DEF_VAR("press", NC_DOUBLE, 1, &obsid, "pressure", "hPa");

```

```

05446 NC_DEF_VAR("lon", NC_DOUBLE, 1, &obsid, "longitude", "degrees_east");
05447 NC_DEF_VAR("lat", NC_DOUBLE, 1, &obsid, "latitude", "degrees_north");
05448 for (int iq = 0; iq < ctl->nq; iq++)
05449     NC_DEF_VAR(ctl->qnt_name[iq], NC_DOUBLE, 1, &obsid,
05450               ctl->qnt_longname[iq], ctl->qnt_unit[iq]);
05451
05452 /* Define global attributes... */
05453 NC_PUT_ATT_GLOBAL("featureType", "point");
05454
05455 /* End definitions... */
05456 NC(nc_enddef(ncid));
05457
05458 /* Write data... */
05459 NC_PUT_DOUBLE("time", atm->time, 0);
05460 NC_PUT_DOUBLE("press", atm->p, 0);
05461 NC_PUT_DOUBLE("lon", atm->lon, 0);
05462 NC_PUT_DOUBLE("lat", atm->lat, 0);
05463 for (int iq = 0; iq < ctl->nq; iq++)
05464     NC_PUT_DOUBLE(ctl->qnt_name[iq], atm->q[iq], 0);
05465
05466 /* Close file... */
05467 NC(nc_close(ncid));
05468 }

```

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

Write CSI data.

Definition at line 5472 of file libtrac.c.

```

05476 {
05477
05478 static FILE *out;
05479
05480 static double *modmean, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
05481             dlon, dlat, dz, x[NCSI], y[NCSI];
05482
05483 static int *obscount, ct, cx, cy, cz, ip, ix, iy, iz, n, nob;
05484
05485 /* Set timer... */
05486 SELECT_TIMER("WRITE_CSI", "OUTPUT", NVTX_WRITE);
05487
05488 /* Init... */
05489 if (t == ctl->t_start) {
05490
05491     /* Check quantity index for mass... */
05492     if (ctl->qnt_m < 0)
05493         ERRMSG("Need quantity mass!");
05494
05495     /* Allocate... */
05496     ALLOC(area, double,
05497           ctl->csi_ny);
05498     ALLOC(rt, double,
05499           NOBS);
05500     ALLOC(rz, double,
05501           NOBS);
05502     ALLOC(rlon, double,
05503           NOBS);
05504     ALLOC(rlat, double,
05505           NOBS);
05506     ALLOC(robs, double,
05507           NOBS);
05508
05509     /* Read observation data... */
05510     read_obs(ctl->csi_obsfile, rt, rz, rlon, rlat, robs, &nob);
05511
05512     /* Create new file... */
05513     LOG(1, "Write CSI data: %s", filename);
05514     if (!(out = fopen(filename, "w")))
05515         ERRMSG("Cannot create file!");
05516
05517     /* Write header... */
05518     fprintf(out,
05519            "# $1 = time [s]\n"

```



```

05520         "# $2 = number of hits (cx)\n"
05521         "# $3 = number of misses (cy)\n"
05522         "# $4 = number of false alarms (cz)\n"
05523         "# $5 = number of observations (cx + cy)\n"
05524         "# $6 = number of forecasts (cx + cz)\n"
05525         "# $7 = bias (ratio of forecasts and observations) [%%]\n"
05526         "# $8 = probability of detection (POD) [%%]\n"
05527         "# $9 = false alarm rate (FAR) [%%]\n"
05528         "# $10 = critical success index (CSI) [%%]\n");
05529     fprintf(out,
05530         "# $11 = hits associated with random chance\n"
05531         "# $12 = equitable threat score (ETS) [%%]\n"
05532         "# $13 = Pearson linear correlation coefficient\n"
05533         "# $14 = Spearman rank-order correlation coefficient\n"
05534         "# $15 = column density mean error (F - O) [kg/m^2]\n"
05535         "# $16 = column density root mean square error (RMSE) [kg/m^2]\n"
05536         "# $17 = column density mean absolute error [kg/m^2]\n"
05537         "# $18 = number of data points\n\n");
05538
05539     /* Set grid box size... */
05540     dz = (ctl->csi_z1 - ctl->csi_z0) / ctl->csi_nz;
05541     dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
05542     dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
05543
05544     /* Set horizontal coordinates... */
05545     for (iy = 0; iy < ctl->csi_ny; iy++) {
05546         double lat = ctl->csi_lat0 + dlat * (iy + 0.5);
05547         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.) * cos(lat * M_PI / 180.);
05548     }
05549 }
05550
05551 /* Set time interval... */
05552 double t0 = t - 0.5 * ctl->dt_mod;
05553 double t1 = t + 0.5 * ctl->dt_mod;
05554
05555 /* Allocate... */
05556 ALLOC(modmean, double,
05557     ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05558 ALLOC(obsmean, double,
05559     ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05560 ALLOC(obscount, int,
05561     ctl->csi_nx * ctl->csi_ny * ctl->csi_nz);
05562
05563 /* Loop over observations... */
05564 for (int i = 0; i < nobs; i++) {
05565
05566     /* Check time... */
05567     if (rt[i] < t0)
05568         continue;
05569     else if (rt[i] >= t1)
05570         break;
05571
05572     /* Check observation data... */
05573     if (!isfinite(robs[i]))
05574         continue;
05575
05576     /* Calculate indices... */
05577     ix = (int) ((rlon[i] - ctl->csi_lon0) / dlon);
05578     iy = (int) ((rlat[i] - ctl->csi_lat0) / dlat);
05579     iz = (int) ((rz[i] - ctl->csi_z0) / dz);
05580
05581     /* Check indices... */
05582     if (ix < 0 || ix >= ctl->csi_nx ||
05583         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
05584         continue;
05585
05586     /* Get mean observation index... */
05587     int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05588     obsmean[idx] += robs[i];
05589     obscount[idx]++;
05590 }
05591
05592 /* Analyze model data... */
05593 for (ip = 0; ip < atm->np; ip++) {
05594
05595     /* Check time... */
05596     if (atm->time[ip] < t0 || atm->time[ip] > t1)
05597         continue;
05598
05599     /* Get indices... */
05600     ix = (int) ((atm->lon[ip] - ctl->csi_lon0) / dlon);
05601     iy = (int) ((atm->lat[ip] - ctl->csi_lat0) / dlat);
05602     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0) / dz);
05603
05604     /* Check indices... */
05605     if (ix < 0 || ix >= ctl->csi_nx ||
05606         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)

```

```

05607         continue;
05608
05609         /* Get total mass in grid cell... */
05610         int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05611         modmean[idx] += atm->q[ctl->qnt_m][ip];
05612     }
05613
05614     /* Analyze all grid cells... */
05615     for (ix = 0; ix < ctl->csi_nx; ix++)
05616         for (iy = 0; iy < ctl->csi_ny; iy++)
05617             for (iz = 0; iz < ctl->csi_nz; iz++) {
05618
05619                 /* Calculate mean observation index... */
05620                 int idx = ARRAY_3D(ix, iy, ctl->csi_ny, iz, ctl->csi_nz);
05621                 if (obscount[idx] > 0)
05622                     obsmean[idx] /= obscount[idx];
05623
05624                 /* Calculate column density... */
05625                 if (modmean[idx] > 0)
05626                     modmean[idx] /= (1e6 * area[iy]);
05627
05628                 /* Calculate CSI... */
05629                 if (obscount[idx] > 0) {
05630                     ct++;
05631                     if (obsmean[idx] >= ctl->csi_obsmin &&
05632                         modmean[idx] >= ctl->csi_modmin)
05633                         cx++;
05634                     else if (obsmean[idx] >= ctl->csi_obsmin &&
05635                             modmean[idx] < ctl->csi_modmin)
05636                         cy++;
05637                     else if (obsmean[idx] < ctl->csi_obsmin &&
05638                             modmean[idx] >= ctl->csi_modmin)
05639                         cz++;
05640                 }
05641
05642                 /* Save data for other verification statistics... */
05643                 if (obscount[idx] > 0
05644                     && (obsmean[idx] >= ctl->csi_obsmin
05645                         || modmean[idx] >= ctl->csi_modmin)) {
05646                     x[n] = modmean[idx];
05647                     y[n] = obsmean[idx];
05648                     if ((++n) > NCSI)
05649                         ERRMSG("Too many data points to calculate statistics!");
05650                 }
05651             }
05652
05653     /* Write output... */
05654     if (fmod(t, ctl->csi_dt_out) == 0) {
05655
05656         /* Calculate verification statistics
05657          (https://www.cawcr.gov.au/projects/verification/) ... */
05658         static double work[2 * NCSI];
05659         int n_obs = cx + cy;
05660         int n_for = cx + cz;
05661         double bias = (n_obs > 0) ? 100. * n_for / n_obs : GSL_NAN;
05662         double pod = (n_obs > 0) ? (100. * cx) / n_obs : GSL_NAN;
05663         double far = (n_for > 0) ? (100. * cz) / n_for : GSL_NAN;
05664         double csi = (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN;
05665         double cx_rd = (ct > 0) ? (1. * n_obs * n_for) / ct : GSL_NAN;
05666         double ets = (cx + cy + cz - cx_rd > 0) ?
05667             (100. * (cx - cx_rd)) / (cx + cy + cz - cx_rd) : GSL_NAN;
05668         double rho_p =
05669             (n > 0) ? gsl_stats_correlation(x, 1, y, 1, (size_t) n) : GSL_NAN;
05670         double rho_s =
05671             (n > 0) ? gsl_stats_spearman(x, 1, y, 1, (size_t) n, work) : GSL_NAN;
05672         for (int i = 0; i < n; i++)
05673             work[i] = x[i] - y[i];
05674         double mean = (n > 0) ? gsl_stats_mean(work, 1, (size_t) n) : GSL_NAN;
05675         double rmse = (n > 0) ? gsl_stats_sd_with_fixed_mean(work, 1, (size_t) n,
05676             0.0) : GSL_NAN;
05677         double absdev =
05678             (n > 0) ? gsl_stats_absdev_m(work, 1, (size_t) n, 0.0) : GSL_NAN;
05679
05680         /* Write... */
05681         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g %g %g %g %g %g %d\n",
05682             t, cx, cy, cz, n_obs, n_for, bias, pod, far, csi, cx_rd, ets,
05683             rho_p, rho_s, mean, rmse, absdev, n);
05684
05685         /* Set counters to zero... */
05686         n = ct = cx = cy = cz = 0;
05687     }
05688
05689     /* Free... */
05690     free(modmean);
05691     free(obsmean);
05692     free(obscount);
05693

```

```

05694  /* Finalize... */
05695  if (t == ctl->t_stop) {
05696
05697      /* Close output file... */
05698      fclose(out);
05699
05700      /* Free... */
05701      free(area);
05702      free(rt);
05703      free(rz);
05704      free(rlon);
05705      free(rlat);
05706      free(robs);
05707  }
05708 }

```

Here is the call graph for this function:



5.23.3.74 write_ens() void write_ens (

```

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

```

Write ensemble data.

Definition at line 5712 of file libtrac.c.

```

05716  {
05717
05718      static FILE *out;
05719
05720      static double dummy, lat, lon, qm[NQ][NENS], qs[NQ][NENS], xm[NENS][3],
05721          x[3], zm[NENS];
05722
05723      static int n[NENS];
05724
05725      /* Set timer... */
05726      SELECT_TIMER("WRITE_ENS", "OUTPUT", NVTX_WRITE);
05727
05728      /* Check quantities... */
05729      if (ctl->qnt_ens < 0)
05730          ERRMSG("Missing ensemble IDs!");
05731
05732      /* Set time interval... */
05733      double t0 = t - 0.5 * ctl->dt_mod;
05734      double t1 = t + 0.5 * ctl->dt_mod;
05735
05736      /* Init... */
05737      for (int i = 0; i < NENS; i++) {
05738          for (int iq = 0; iq < ctl->nq; iq++)
05739              qm[iq][i] = qs[iq][i] = 0;
05740          xm[i][0] = xm[i][1] = xm[i][2] = zm[i] = 0;
05741          n[i] = 0;
05742      }
05743
05744      /* Loop over air parcels... */
05745      for (int ip = 0; ip < atm->np; ip++) {
05746
05747          /* Check time... */
05748          if (atm->time[ip] < t0 || atm->time[ip] > t1)

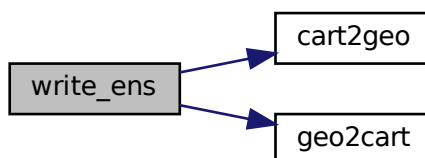
```

```

05749         continue;
05750
05751         /* Check ensemble ID... */
05752         if (atm->q[ctl->qnt_ens][ip] < 0 || atm->q[ctl->qnt_ens][ip] >= NENS)
05753             ERRMSG("Ensemble ID is out of range!");
05754
05755         /* Get means... */
05756         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
05757         for (int iq = 0; iq < ctl->nq; iq++) {
05758             qm[iq][ctl->qnt_ens] += atm->q[iq][ip];
05759             qs[iq][ctl->qnt_ens] += SQR(atm->q[iq][ip]);
05760         }
05761         xm[ctl->qnt_ens][0] += x[0];
05762         xm[ctl->qnt_ens][1] += x[1];
05763         xm[ctl->qnt_ens][2] += x[2];
05764         zm[ctl->qnt_ens] += Z(atm->p[ip]);
05765         n[ctl->qnt_ens]++;
05766     }
05767
05768     /* Create file... */
05769     LOG(1, "Write ensemble data: %s", filename);
05770     if (!out = fopen(filename, "w"))
05771         ERRMSG("Cannot create file!");
05772
05773     /* Write header... */
05774     fprintf(out,
05775             "# $1 = time [s]\n"
05776             "# $2 = altitude [km]\n"
05777             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
05778     for (int iq = 0; iq < ctl->nq; iq++)
05779         fprintf(out, "# $qd = %s (mean) [%s]\n", 5 + iq,
05780                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05781     for (int iq = 0; iq < ctl->nq; iq++)
05782         fprintf(out, "# $qd = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
05783                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
05784     fprintf(out, "# $qd = number of members\n\n", 5 + 2 * ctl->nq);
05785
05786     /* Write data... */
05787     for (int i = 0; i < NENS; i++)
05788         if (n[i] > 0) {
05789             cart2geo(xm[i], &dummy, &lon, &lat);
05790             fprintf(out, "%.2f %g %g %g", t, zm[i] / n[i], lon, lat);
05791             for (int iq = 0; iq < ctl->nq; iq++) {
05792                 fprintf(out, " ");
05793                 fprintf(out, ctl->qnt_format[iq], qm[iq][i] / n[i]);
05794             }
05795             for (int iq = 0; iq < ctl->nq; iq++) {
05796                 fprintf(out, " ");
05797                 double var = qs[iq][i] / n[i] - SQR(qm[iq][i] / n[i]);
05798                 fprintf(out, ctl->qnt_format[iq], (var > 0 ? sqrt(var) : 0));
05799             }
05800             fprintf(out, " %d\n", n[i]);
05801         }
05802
05803     /* Close file... */
05804     fclose(out);
05805 }

```

Here is the call graph for this function:



5.23.3.75 write_grid() void write_grid (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )
```

Write gridded data.

Definition at line 5809 of file libtrac.c.

```

5815     {
5816
5817     double *cd, *mass, *vmr_expl, *vmr_impl, *z, *lon, *lat, *area, *press;
5818
5819     int *ixs, *iys, *izs, *np;
5820
5821     /* Set timer... */
5822     SELECT_TIMER("WRITE_GRID", "OUTPUT", NVTX_WRITE);
5823
5824     /* Write info... */
5825     LOG(1, "Write grid data: %s", filename);
5826
5827     /* Allocate... */
5828     ALLOC(cd, double,
5829           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5830     ALLOC(mass, double,
5831           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5832     ALLOC(vmr_expl, double,
5833           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5834     ALLOC(vmr_impl, double,
5835           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5836     ALLOC(z, double,
5837           ctl->grid_nz);
5838     ALLOC(lon, double,
5839           ctl->grid_nx);
5840     ALLOC(lat, double,
5841           ctl->grid_ny);
5842     ALLOC(area, double,
5843           ctl->grid_ny);
5844     ALLOC(press, double,
5845           ctl->grid_nz);
5846     ALLOC(np, int,
5847           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
5848     ALLOC(ixs, int,
5849           atm->np);
5850     ALLOC(iys, int,
5851           atm->np);
5852     ALLOC(izs, int,
5853           atm->np);
5854
5855     /* Set grid box size... */
5856     double dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
5857     double dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
5858     double dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
5859
5860     /* Set vertical coordinates... */
5861     #pragma omp parallel for default(shared)
5862     for (int iz = 0; iz < ctl->grid_nz; iz++) {
5863         z[iz] = ctl->grid_z0 + dz * (iz + 0.5);
5864         press[iz] = P(z[iz]);
5865     }
5866
5867     /* Set horizontal coordinates... */
5868     for (int ix = 0; ix < ctl->grid_nx; ix++)
5869         lon[ix] = ctl->grid_lon0 + dlon * (ix + 0.5);
5870     #pragma omp parallel for default(shared)
5871     for (int iy = 0; iy < ctl->grid_ny; iy++) {
5872         lat[iy] = ctl->grid_lat0 + dlat * (iy + 0.5);
5873         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
5874             * cos(lat[iy] * M_PI / 180.);
5875     }
5876
5877     /* Set time interval for output... */
5878     double t0 = t - 0.5 * ctl->dt_mod;
5879     double t1 = t + 0.5 * ctl->dt_mod;
5880
5881     /* Get grid box indices... */
5882     #pragma omp parallel for default(shared)
5883     for (int ip = 0; ip < atm->np; ip++) {
5884         ixs[ip] = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
5885         iys[ip] = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
```

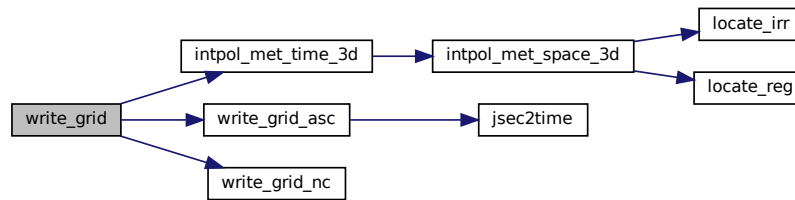
```

05886     izes[ip] = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
05887     if (atm->time[ip] < t0 || atm->time[ip] > t1
05888         || ix[ip] < 0 || ix[ip] >= ctl->grid_nx
05889         || iy[ip] < 0 || iy[ip] >= ctl->grid_ny
05890         || iz[ip] < 0 || iz[ip] >= ctl->grid_nz)
05891         izes[ip] = -1;
05892 }
05893
05894 /* Average data... */
05895 for (int ip = 0; ip < atm->np; ip++)
05896     if (izes[ip] >= 0) {
05897         int idx =
05898             ARRAY_3D(ix[ip], iy[ip], ctl->grid_ny, iz[ip], ctl->grid_nz);
05899         np[idx]++;
05900         if (ctl->qnt_m >= 0)
05901             mass[idx] += atm->q[ctl->qnt_m][ip];
05902         if (ctl->qnt_vmr >= 0)
05903             vmr_expl[idx] += atm->q[ctl->qnt_vmr][ip];
05904     }
05905
05906 /* Calculate column density and vmr... */
05907 #pragma omp parallel for default(shared)
05908 for (int ix = 0; ix < ctl->grid_nx; ix++)
05909     for (int iy = 0; iy < ctl->grid_ny; iy++)
05910         for (int iz = 0; iz < ctl->grid_nz; iz++) {
05911
05912             /* Get grid index... */
05913             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
05914
05915             /* Calculate column density... */
05916             cd[idx] = GSL_NAN;
05917             if (ctl->qnt_m >= 0)
05918                 cd[idx] = mass[idx] / (1e6 * area[iy]);
05919
05920             /* Calculate volume mixing ratio (implicit)... */
05921             vmr_impl[idx] = GSL_NAN;
05922             if (ctl->qnt_m >= 0 && ctl->molmass > 0) {
05923                 vmr_impl[idx] = 0;
05924                 if (mass[idx] > 0) {
05925
05926                     /* Get temperature... */
05927                     double temp;
05928                     INTPOL_INIT;
05929                     intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
05930                                     lon[ix], lat[iy], &temp, ci, cw, 1);
05931
05932                     /* Calculate volume mixing ratio... */
05933                     vmr_impl[idx] = MA / ctl->molmass * mass[idx]
05934                                     / (RHO(press[iz], temp) * 1e6 * area[iy] * 1e3 * dz);
05935                 }
05936             }
05937
05938             /* Calculate volume mixing ratio (explicit)... */
05939             if (ctl->qnt_vmr >= 0 && np[idx] > 0)
05940                 vmr_expl[idx] /= np[idx];
05941             else
05942                 vmr_expl[idx] = GSL_NAN;
05943         }
05944
05945 /* Write ASCII data... */
05946 if (ctl->grid_type == 0)
05947     write_grid_asc(filename, ctl, cd, vmr_expl, vmr_impl,
05948                   t, z, lon, lat, area, dz, np);
05949
05950 /* Write netCDF data... */
05951 else if (ctl->grid_type == 1)
05952     write_grid_nc(filename, ctl, cd, vmr_expl, vmr_impl,
05953                  t, z, lon, lat, area, dz, np);
05954
05955 /* Error message... */
05956 else
05957     ERRMSG("Grid data format GRID_TYPE unknown!");
05958
05959 /* Free... */
05960 free(cd);
05961 free(mass);
05962 free(vmr_expl);
05963 free(vmr_impl);
05964 free(z);
05965 free(lon);
05966 free(lat);
05967 free(area);
05968 free(press);
05969 free(np);
05970 free(ixs);
05971 free(iys);
05972 free(izes);

```

```
05973 }
```

Here is the call graph for this function:



5.23.3.76 write_grid_asc() void write_grid_asc (

```

    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )

```

Write gridded data in ASCII format.

Definition at line 5977 of file libtrac.c.

```

05989     {
05990
05991     FILE *in, *out;
05992
05993     char line[LEN];
05994
05995     /* Check if gnuplot output is requested... */
05996     if (ctl->grid_gpfile[0] != '-') {
05997
05998         /* Create gnuplot pipe... */
05999         if (!(out = popen("gnuplot", "w")))
06000             ERRMSG("Cannot create pipe to gnuplot!");
06001
06002         /* Set plot filename... */
06003         fprintf(out, "set out \"%s.png\"\n", filename);
06004
06005         /* Set time string... */
06006         double r;
06007         int year, mon, day, hour, min, sec;
06008         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
06009         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
06010             year, mon, day, hour, min);
06011
06012         /* Dump gnuplot file to pipe... */
06013         if (!(in = fopen(ctl->grid_gpfile, "r")))
06014             ERRMSG("Cannot open file!");
06015         while (fgets(line, LEN, in))
06016             fprintf(out, "%s", line);
06017         fclose(in);
06018     }
06019
06020     else {

```

```

06021
06022     /* Create file... */
06023     if (!out = fopen(filename, "w"))
06024         ERRMSG("Cannot create file!");
06025 }
06026
06027 /* Write header... */
06028 fprintf(out,
06029     "# $1 = time [s]\n"
06030     "# $2 = altitude [km]\n"
06031     "# $3 = longitude [deg]\n"
06032     "# $4 = latitude [deg]\n"
06033     "# $5 = surface area [km^2]\n"
06034     "# $6 = layer depth [km]\n"
06035     "# $7 = number of particles [l]\n"
06036     "# $8 = column density (implicit) [kg/m^2]\n"
06037     "# $9 = volume mixing ratio (implicit) [ppv]\n"
06038     "# $10 = volume mixing ratio (explicit) [ppv]\n\n");
06039
06040 /* Write data... */
06041 for (int ix = 0; ix < ctl->grid_nx; ix++) {
06042     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
06043         fprintf(out, "\n");
06044     for (int iy = 0; iy < ctl->grid_ny; iy++) {
06045         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
06046             fprintf(out, "\n");
06047         for (int iz = 0; iz < ctl->grid_nz; iz++) {
06048             int idx = ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz);
06049             if (!ctl->grid_sparse || vmr_expl[idx] > 0 || vmr_impl[idx] > 0)
06050                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n",
06051                     t, z[iz], lon[ix], lat[iy], area[iy], dz,
06052                     np[idx], cd[idx], vmr_impl[idx], vmr_expl[idx]);
06053         }
06054     }
06055 }
06056
06057 /* Close file... */
06058 fclose(out);
06059 }

```

Here is the call graph for this function:



5.23.3.77 write_grid_nc() void write_grid_nc (

```

    const char * filename,
    ctl_t * ctl,
    double * cd,
    double * vmr_expl,
    double * vmr_impl,
    double t,
    double * z,
    double * lon,
    double * lat,
    double * area,
    double dz,
    int * np )

```

Write gridded data in netCDF format.

Definition at line 6063 of file libtrac.c.

```

60675     {
60676
60677     double *help;
60678
60679     int *help2, ncid, dimid[10], varid;
60680
60681     size_t start[2], count[2];
60682
60683     /* Allocate... */
60684     ALLOC(help, double,
60685           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
60686     ALLOC(help2, int,
60687           ctl->grid_nx * ctl->grid_ny * ctl->grid_nz);
60688
60689     /* Create file... */
60690     nc_create(filename, NC_CLOBBER, &ncid);
60691
60692     /* Define dimensions... */
60693     NC(nc_def_dim(ncid, "time", 1, &dimid[0]));
60694     NC(nc_def_dim(ncid, "z", (size_t) ctl->grid_nz, &dimid[1]));
60695     NC(nc_def_dim(ncid, "lat", (size_t) ctl->grid_ny, &dimid[2]));
60696     NC(nc_def_dim(ncid, "lon", (size_t) ctl->grid_nx, &dimid[3]));
60697     NC(nc_def_dim(ncid, "dz", 1, &dimid[4]));
60698
60699     /* Define variables and their attributes... */
60700     NC_DEF_VAR("time", NC_DOUBLE, 1, &dimid[0], "time",
60701              "seconds since 2000-01-01 00:00:00 UTC");
60702     NC_DEF_VAR("z", NC_DOUBLE, 1, &dimid[1], "altitude", "km");
60703     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dimid[2], "latitude", "degrees_north");
60704     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dimid[3], "longitude", "degrees_east");
60705     NC_DEF_VAR("dz", NC_DOUBLE, 1, &dimid[4], "layer depth", "km");
60706     NC_DEF_VAR("area", NC_DOUBLE, 1, &dimid[2], "surface area", "km**2");
60707     NC_DEF_VAR("cd", NC_FLOAT, 4, dimid, "column density", "kg m**-2");
60708     NC_DEF_VAR("vmr_impl", NC_FLOAT, 4, dimid,
60709              "volume mixing ratio (implicit)", "ppv");
60710     NC_DEF_VAR("vmr_expl", NC_FLOAT, 4, dimid,
60711              "volume mixing ratio (explicit)", "ppv");
60712     NC_DEF_VAR("np", NC_INT, 4, dimid, "number of particles", "1");
60713
60714     /* End definitions... */
60715     NC(nc_enddef(ncid));
60716
60717     /* Write data... */
60718     NC_PUT_DOUBLE("time", &t, 0);
60719     NC_PUT_DOUBLE("lon", lon, 0);
60720     NC_PUT_DOUBLE("lat", lat, 0);
60721     NC_PUT_DOUBLE("z", z, 0);
60722     NC_PUT_DOUBLE("area", area, 0);
60723     NC_PUT_DOUBLE("dz", &dz, 0);
60724
60725     for (int ix = 0; ix < ctl->grid_nx; ix++)
60726         for (int iy = 0; iy < ctl->grid_ny; iy++)
60727             for (int iz = 0; iz < ctl->grid_nz; iz++)
60728                 help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60729                     cd[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60730     NC_PUT_DOUBLE("cd", help, 0);
60731
60732     for (int ix = 0; ix < ctl->grid_nx; ix++)
60733         for (int iy = 0; iy < ctl->grid_ny; iy++)
60734             for (int iz = 0; iz < ctl->grid_nz; iz++)
60735                 help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60736                     vmr_impl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60737     NC_PUT_DOUBLE("vmr_impl", help, 0);
60738
60739     for (int ix = 0; ix < ctl->grid_nx; ix++)
60740         for (int iy = 0; iy < ctl->grid_ny; iy++)
60741             for (int iz = 0; iz < ctl->grid_nz; iz++)
60742                 help[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60743                     vmr_expl[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60744     NC_PUT_DOUBLE("vmr_expl", help, 0);
60745
60746     for (int ix = 0; ix < ctl->grid_nx; ix++)
60747         for (int iy = 0; iy < ctl->grid_ny; iy++)
60748             for (int iz = 0; iz < ctl->grid_nz; iz++)
60749                 help2[ARRAY_3D(iz, iy, ctl->grid_ny, ix, ctl->grid_nx)] =
60750                     np[ARRAY_3D(ix, iy, ctl->grid_ny, iz, ctl->grid_nz)];
60751     NC_PUT_INT("np", help2, 0);
60752
60753     /* Close file... */
60754     NC(nc_close(ncid));
60755
60756     /* Free... */
60757     free(help);
60758     free(help2);
60759 }

```

```

5.23.3.78 write_met() int write_met (
    char * filename,
    ctl_t * ctl,
    met_t * met )

```

Read meteo data file.

Definition at line 6163 of file libtrac.c.

```

6166         {
6167
6168         /* Set timer... */
6169         SELECT_TIMER("WRITE_MET", "OUTPUT", NVTX_WRITE);
6170
6171         /* Write info... */
6172         LOG(1, "Write meteo data: %s", filename);
6173
6174         /* Check compression flags... */
6175 #ifndef ZFP
6176         if (ctl->met_type == 3)
6177             ERRMSG("zfp compression not supported!");
6178 #endif
6179 #ifndef ZSTD
6180         if (ctl->met_type == 4)
6181             ERRMSG("zstd compression not supported!");
6182 #endif
6183
6184         /* Write binary... */
6185         if (ctl->met_type >= 1 && ctl->met_type <= 4) {
6186
6187             /* Create file... */
6188             FILE *out;
6189             if (!(out = fopen(filename, "w")))
6190                 ERRMSG("Cannot create file!");
6191
6192             /* Write type of binary data... */
6193             FWRITE(&ctl->met_type, int,
6194                 1,
6195                 out);
6196
6197             /* Write version of binary data... */
6198             int version = 100;
6199             FWRITE(&version, int,
6200                 1,
6201                 out);
6202
6203             /* Write grid data... */
6204             FWRITE(&met->time, double,
6205                 1,
6206                 out);
6207             FWRITE(&met->nx, int,
6208                 1,
6209                 out);
6210             FWRITE(&met->ny, int,
6211                 1,
6212                 out);
6213             FWRITE(&met->np, int,
6214                 1,
6215                 out);
6216             FWRITE(met->lon, double,
6217                 (size_t) met->nx,
6218                 out);
6219             FWRITE(met->lat, double,
6220                 (size_t) met->ny,
6221                 out);
6222             FWRITE(met->p, double,
6223                 (size_t) met->np,
6224                 out);
6225
6226             /* Write surface data... */
6227             write_met_bin_2d(out, met, met->ps, "PS");
6228             write_met_bin_2d(out, met, met->ts, "TS");
6229             write_met_bin_2d(out, met, met->zs, "ZS");
6230             write_met_bin_2d(out, met, met->us, "US");
6231             write_met_bin_2d(out, met, met->vs, "VS");
6232             write_met_bin_2d(out, met, met->pbl, "PBL");
6233             write_met_bin_2d(out, met, met->pt, "PT");
6234             write_met_bin_2d(out, met, met->tt, "TT");
6235             write_met_bin_2d(out, met, met->zt, "ZT");
6236             write_met_bin_2d(out, met, met->h2ot, "H2OT");

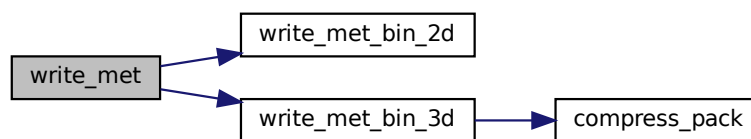
```

```

06237     write_met_bin_2d(out, met, met->pct, "PCT");
06238     write_met_bin_2d(out, met, met->pcb, "PCB");
06239     write_met_bin_2d(out, met, met->cl, "CL");
06240     write_met_bin_2d(out, met, met->plcl, "PLCL");
06241     write_met_bin_2d(out, met, met->plfc, "PLFC");
06242     write_met_bin_2d(out, met, met->pel, "PEL");
06243     write_met_bin_2d(out, met, met->cape, "CAPE");
06244     write_met_bin_2d(out, met, met->cin, "CIN");
06245
06246     /* Write level data... */
06247     write_met_bin_3d(out, ctl, met, met->z, "Z", 0, 0.5);
06248     write_met_bin_3d(out, ctl, met, met->t, "T", 0, 5.0);
06249     write_met_bin_3d(out, ctl, met, met->u, "U", 8, 0);
06250     write_met_bin_3d(out, ctl, met, met->v, "V", 8, 0);
06251     write_met_bin_3d(out, ctl, met, met->w, "W", 8, 0);
06252     write_met_bin_3d(out, ctl, met, met->pv, "PV", 8, 0);
06253     write_met_bin_3d(out, ctl, met, met->h2o, "H2O", 8, 0);
06254     write_met_bin_3d(out, ctl, met, met->o3, "O3", 8, 0);
06255     write_met_bin_3d(out, ctl, met, met->lwc, "LWC", 8, 0);
06256     write_met_bin_3d(out, ctl, met, met->iwc, "IWC", 8, 0);
06257
06258     /* Write final flag... */
06259     int final = 999;
06260     FWRITE(&final, int,
06261           1,
06262           out);
06263
06264     /* Close file... */
06265     fclose(out);
06266 }
06267
06268     return 0;
06269 }

```

Here is the call graph for this function:



5.23.3.79 write_met_bin_2d() void write_met_bin_2d (

```

FILE * out,
met_t * met,
float var[EX][EY],
char * varname )

```

Write 2-D meteo variable.

Definition at line 6273 of file libtrac.c.

```

06277     {
06278
06279     float *help;
06280
06281     /* Allocate... */
06282     ALLOC(help, float,
06283           EX * EY);
06284
06285     /* Copy data... */
06286     for (int ix = 0; ix < met->nx; ix++)
06287         for (int iy = 0; iy < met->ny; iy++)
06288             help[ARRAY_2D(ix, iy, met->ny)] = var[ix][iy];

```

```

06289
06290 /* Write uncompressed data... */
06291 LOG(2, "Write 2-D variable: %s (uncompressed)", varname);
06292 FWRITE(help, float,
06293         (size_t) (met->nx * met->ny),
06294         out);
06295
06296 /* Free... */
06297 free(help);
06298 }

```

5.23.3.80 write_met_bin_3d() void write_met_bin_3d (

```

FILE * out,
ctl_t * ctl,
met_t * met,
float var[EX][EY][EP],
char * varname,
int precision,
double tolerance )

```

Write 3-D meteo variable.

Definition at line 6302 of file libtrac.c.

```

06309 {
06310
06311     float *help;
06312
06313     /* Allocate... */
06314     ALLOC(help, float,
06315           EX * EY * EP);
06316
06317     /* Copy data... */
06318 #pragma omp parallel for default(shared) collapse(2)
06319     for (int ix = 0; ix < met->nx; ix++)
06320         for (int iy = 0; iy < met->ny; iy++)
06321             for (int ip = 0; ip < met->np; ip++)
06322                 help[ARRAY_3D(ix, iy, met->ny, ip, met->np)] = var[ix][iy][ip];
06323
06324     /* Write uncompressed data... */
06325     if (ctl->met_type == 1) {
06326         LOG(2, "Write 3-D variable: %s (uncompressed)", varname);
06327         FWRITE(help, float,
06328               (size_t) (met->nx * met->ny * met->np),
06329               out);
06330     }
06331
06332     /* Write packed data... */
06333     else if (ctl->met_type == 2)
06334         compress_pack(varname, help, (size_t) (met->ny * met->nx),
06335                      (size_t) met->np, 0, out);
06336
06337     /* Write zfp data... */
06338 #ifdef ZFP
06339     else if (ctl->met_type == 3)
06340         compress_zfp(varname, help, met->np, met->ny, met->nx, precision,
06341                     tolerance, 0, out);
06342 #endif
06343
06344     /* Write zstd data... */
06345 #ifdef ZSTD
06346     else if (ctl->met_type == 4)
06347         compress_zstd(varname, help, (size_t) (met->np * met->ny * met->nx), 0,
06348                      out);
06349 #endif
06350
06351     /* Unknown method... */
06352     else {
06353         ERRMSG("MET_TYPE not supported!");
06354         LOG(3, "%d %g", precision, tolerance);
06355     }
06356
06357     /* Free... */
06358     free(help);
06359 }

```

Here is the call graph for this function:



5.23.3.81 write_prof() void write_prof (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write profile data.

Definition at line 6363 of file libtrac.c.

```

06369     {
06370
06371     static FILE *out;
06372
06373     static double *mass, *obsmean, *rt, *rz, *rlon, *rlat, *robs, *area,
06374         dz, dlon, dlat, *lon, *lat, *z, *press, temp, vmr, h2o, o3;
06375
06376     static int nob, *obscount, ip, okay;
06377
06378     /* Set timer... */
06379     SELECT_TIMER("WRITE_PROF", "OUTPUT", NVTX_WRITE);
06380
06381     /* Init... */
06382     if (t == ctl->t_start) {
06383
06384         /* Check quantity index for mass... */
06385         if (ctl->qnt_m < 0)
06386             ERRMSG("Need quantity mass!");
06387
06388         /* Check molar mass... */
06389         if (ctl->molmass <= 0)
06390             ERRMSG("Specify molar mass!");
06391
06392         /* Allocate... */
06393         ALLOC(lon, double,
06394             ctl->prof_nx);
06395         ALLOC(lat, double,
06396             ctl->prof_ny);
06397         ALLOC(area, double,
06398             ctl->prof_ny);
06399         ALLOC(z, double,
06400             ctl->prof_nz);
06401         ALLOC(press, double,
06402             ctl->prof_nz);
06403         ALLOC(rt, double,
06404             NOBS);
06405         ALLOC(rz, double,
06406             NOBS);
06407         ALLOC(rlon, double,
06408             NOBS);
06409         ALLOC(rlat, double,
06410             NOBS);
06411         ALLOC(robs, double,
06412             NOBS);
06413
06414         /* Read observation data... */

```

```

06415     read_obs(ctl->prof_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06416
06417     /* Create new output file... */
06418     LOG(1, "Write profile data: %s", filename);
06419     if (!out = fopen(filename, "w"))
06420         ERRMSG("Cannot create file!");
06421
06422     /* Write header... */
06423     fprintf(out,
06424             "# $1 = time [s]\n"
06425             "# $2 = altitude [km]\n"
06426             "# $3 = longitude [deg]\n"
06427             "# $4 = latitude [deg]\n"
06428             "# $5 = pressure [hPa]\n"
06429             "# $6 = temperature [K]\n"
06430             "# $7 = volume mixing ratio [ppv]\n"
06431             "# $8 = H2O volume mixing ratio [ppv]\n"
06432             "# $9 = O3 volume mixing ratio [ppv]\n"
06433             "# $10 = observed BT index [K]\n"
06434             "# $11 = number of observations\n");
06435
06436     /* Set grid box size... */
06437     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
06438     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
06439     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
06440
06441     /* Set vertical coordinates... */
06442     for (int iz = 0; iz < ctl->prof_nz; iz++) {
06443         z[iz] = ctl->prof_z0 + dz * (iz + 0.5);
06444         press[iz] = P(z[iz]);
06445     }
06446
06447     /* Set horizontal coordinates... */
06448     for (int ix = 0; ix < ctl->prof_nx; ix++)
06449         lon[ix] = ctl->prof_lon0 + dlon * (ix + 0.5);
06450     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06451         lat[iy] = ctl->prof_lat0 + dlat * (iy + 0.5);
06452         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
06453             * cos(lat[iy] * M_PI / 180.);
06454     }
06455 }
06456
06457 /* Set time interval... */
06458 double t0 = t - 0.5 * ctl->dt_mod;
06459 double t1 = t + 0.5 * ctl->dt_mod;
06460
06461 /* Allocate... */
06462 ALLOC(mass, double,
06463       ctl->prof_nx * ctl->prof_ny * ctl->prof_nz);
06464 ALLOC(obsmean, double,
06465       ctl->prof_nx * ctl->prof_ny);
06466 ALLOC(obscount, int,
06467       ctl->prof_nx * ctl->prof_ny);
06468
06469 /* Loop over observations... */
06470 for (int i = 0; i < nobs; i++) {
06471
06472     /* Check time... */
06473     if (rt[i] < t0)
06474         continue;
06475     else if (rt[i] >= t1)
06476         break;
06477
06478     /* Check observation data... */
06479     if (!isfinite(robs[i]))
06480         continue;
06481
06482     /* Calculate indices... */
06483     int ix = (int) ((rlon[i] - ctl->prof_lon0) / dlon);
06484     int iy = (int) ((rlat[i] - ctl->prof_lat0) / dlat);
06485
06486     /* Check indices... */
06487     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
06488         continue;
06489
06490     /* Get mean observation index... */
06491     int idx = ARRAY_2D(ix, iy, ctl->prof_ny);
06492     obsmean[idx] += robs[i];
06493     obscount[idx]++;
06494 }
06495
06496 /* Analyze model data... */
06497 for (ip = 0; ip < atm->np; ip++) {
06498
06499     /* Check time... */
06500     if (atm->time[ip] < t0 || atm->time[ip] > t1)
06501         continue;

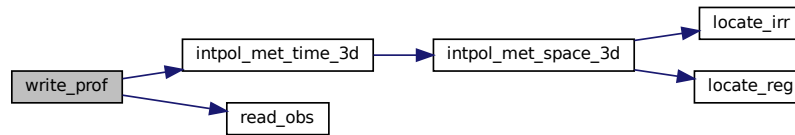
```

```

06502
06503 /* Get indices... */
06504 int ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
06505 int iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
06506 int iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
06507
06508 /* Check indices... */
06509 if (ix < 0 || ix >= ctl->prof_nx ||
06510     iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
06511     continue;
06512
06513 /* Get total mass in grid cell... */
06514 int idx = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06515 mass[idx] += atm->q[ctl->qnt_m][ip];
06516 }
06517
06518 /* Extract profiles... */
06519 for (int ix = 0; ix < ctl->prof_nx; ix++)
06520     for (int iy = 0; iy < ctl->prof_ny; iy++) {
06521         int idx2 = ARRAY_2D(ix, iy, ctl->prof_ny);
06522         if (obscount[idx2] > 0) {
06523
06524             /* Check profile... */
06525             okay = 0;
06526             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06527                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06528                 if (mass[idx3] > 0) {
06529                     okay = 1;
06530                     break;
06531                 }
06532             }
06533             if (!okay)
06534                 continue;
06535
06536             /* Write output... */
06537             fprintf(out, "\n");
06538
06539             /* Loop over altitudes... */
06540             for (int iz = 0; iz < ctl->prof_nz; iz++) {
06541
06542                 /* Get temperature, water vapor, and ozone... */
06543                 INTPOL_INIT;
06544                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[iz],
06545                                     lon[ix], lat[iy], &temp, ci, cw, 1);
06546                 intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, t, press[iz],
06547                                     lon[ix], lat[iy], &h2o, ci, cw, 0);
06548                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press[iz],
06549                                     lon[ix], lat[iy], &o3, ci, cw, 0);
06550
06551                 /* Calculate volume mixing ratio... */
06552                 int idx3 = ARRAY_3D(ix, iy, ctl->prof_ny, iz, ctl->prof_nz);
06553                 vmr = MA / ctl->molmass * mass[idx3]
06554                     / (RHO(press[iz], temp) * area[iy] * dz * 1e9);
06555
06556                 /* Write output... */
06557                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
06558                         t, z[iz], lon[ix], lat[iy], press[iz], temp, vmr, h2o, o3,
06559                         obsmean[idx2] / obscount[idx2], obscount[idx2]);
06560             }
06561         }
06562     }
06563
06564 /* Free... */
06565 free(mass);
06566 free(obsmean);
06567 free(obscount);
06568
06569 /* Finalize... */
06570 if (t == ctl->t_stop) {
06571
06572     /* Close output file... */
06573     fclose(out);
06574
06575     /* Free... */
06576     free(lon);
06577     free(lat);
06578     free(area);
06579     free(z);
06580     free(press);
06581     free(rt);
06582     free(rz);
06583     free(rlon);
06584     free(rlat);
06585     free(robs);
06586 }
06587 }

```

Here is the call graph for this function:



5.23.3.82 write_sample() void write_sample (

```

    const char * filename,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write sample data.

Definition at line 6591 of file libtrac.c.

```

06597     {
06598
06599     static FILE *out;
06600
06601     static double area, dlat, rmax2, *rt, *rz, *rlon, *rlat, *robs;
06602
06603     static int nobs;
06604
06605     /* Set timer... */
06606     SELECT_TIMER("WRITE_SAMPLE", "OUTPUT", NVTX_WRITE);
06607
06608     /* Init... */
06609     if (t == ctl->t_start) {
06610
06611         /* Allocate... */
06612         ALLOC(rt, double,
06613             NOBS);
06614         ALLOC(rz, double,
06615             NOBS);
06616         ALLOC(rlon, double,
06617             NOBS);
06618         ALLOC(rlat, double,
06619             NOBS);
06620         ALLOC(robs, double,
06621             NOBS);
06622
06623         /* Read observation data... */
06624         read_obs(ctl->sample_obsfile, rt, rz, rlon, rlat, robs, &nobs);
06625
06626         /* Create output file... */
06627         LOG(1, "Write sample data: %s", filename);
06628         if (!(out = fopen(filename, "w")))
06629             ERRMSG("Cannot create file!");
06630
06631         /* Write header... */
06632         fprintf(out,
06633             "# $1 = time [s]\n"
06634             "# $2 = altitude [km]\n"
06635             "# $3 = longitude [deg]\n"
06636             "# $4 = latitude [deg]\n"
06637             "# $5 = surface area [km^2]\n"
06638             "# $6 = layer depth [km]\n"
06639             "# $7 = number of particles [l]\n"
06640             "# $8 = column density [kg/m^2]\n"
06641             "# $9 = volume mixing ratio [ppv]\n"
06642             "# $10 = observed BT index [K]\n\n");
06643

```



```

06644     /* Set latitude range, squared radius, and area... */
06645     dlat = DY2DEG(ctl->sample_dx);
06646     rmax2 = SQR(ctl->sample_dx);
06647     area = M_PI * rmax2;
06648 }
06649
06650 /* Set time interval for output... */
06651 double t0 = t - 0.5 * ctl->dt_mod;
06652 double t1 = t + 0.5 * ctl->dt_mod;
06653
06654 /* Loop over observations... */
06655 for (int i = 0; i < nob; i++) {
06656
06657     /* Check time... */
06658     if (rt[i] < t0)
06659         continue;
06660     else if (rt[i] >= t1)
06661         break;
06662
06663     /* Calculate Cartesian coordinates... */
06664     double x0[3];
06665     geo2cart(0, rlon[i], rlat[i], x0);
06666
06667     /* Set pressure range... */
06668     double rp = P(rz[i]);
06669     double ptop = P(rz[i] + ctl->sample_dz);
06670     double pbot = P(rz[i] - ctl->sample_dz);
06671
06672     /* Init... */
06673     double mass = 0;
06674     int np = 0;
06675
06676     /* Loop over air parcels... */
06677 #pragma omp parallel for default(shared) reduction(+:mass,np)
06678     for (int ip = 0; ip < atm->np; ip++) {
06679
06680         /* Check time... */
06681         if (atm->time[ip] < t0 || atm->time[ip] > t1)
06682             continue;
06683
06684         /* Check latitude... */
06685         if (fabs(rlat[i] - atm->lat[ip]) > dlat)
06686             continue;
06687
06688         /* Check horizontal distance... */
06689         double x1[3];
06690         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06691         if (DIST2(x0, x1) > rmax2)
06692             continue;
06693
06694         /* Check pressure... */
06695         if (ctl->sample_dz > 0)
06696             if (atm->p[ip] > pbot || atm->p[ip] < ptop)
06697                 continue;
06698
06699         /* Add mass... */
07000         if (ctl->qnt_m >= 0)
07001             mass += atm->q[ctl->qnt_m][ip];
07002         np++;
07003     }
07004
07005     /* Calculate column density... */
07006     double cd = mass / (1e6 * area);
07007
07008     /* Calculate volume mixing ratio... */
07009     double vmr = 0;
07010     if (ctl->molmass > 0 && ctl->sample_dz > 0) {
07011         if (mass > 0) {
07012
07013             /* Get temperature... */
07014             double temp;
07015             INTPOL_INIT;
07016             intpol_met_time_3d(met0, met0->t, met1, met1->t, rt[i], rp,
07017                             rlon[i], rlat[i], &temp, ci, cw, 1);
07018
07019             /* Calculate volume mixing ratio... */
07020             vmr = MA / ctl->molmass * mass
07021                 / (RHO(rp, temp) * 1e6 * area * 1e3 * ctl->sample_dz);
07022         }
07023     } else
07024         vmr = GSL_NAN;
07025
07026     /* Write output... */
07027     fprintf(out, "%.2f %g %g %g %g %g %d %g %g %g\n", rt[i], rz[i],
07028             rlon[i], rlat[i], area, ctl->sample_dz, np, cd, vmr, robs[i]);
07029 }
07030

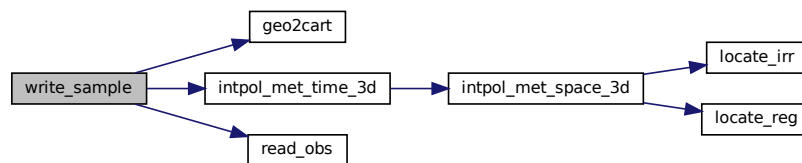
```

```

06731  /* Finalize..... */
06732  if (t == ctl->t_stop) {
06733
06734      /* Close output file... */
06735      fclose(out);
06736
06737      /* Free... */
06738      free(rt);
06739      free(rz);
06740      free(rlon);
06741      free(rlat);
06742      free(robs);
06743  }
06744 }

```

Here is the call graph for this function:



5.23.3.83 write_station() void write_station (

```

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

```

Write station data.

Definition at line 6748 of file libtrac.c.

```

06752  {
06753
06754      static FILE *out;
06755
06756      static double rmax2, x0[3], x1[3];
06757
06758      /* Set timer... */
06759      SELECT_TIMER("WRITE_STATION", "OUTPUT", NVTX_WRITE);
06760
06761      /* Init... */
06762      if (t == ctl->t_start) {
06763
06764          /* Write info... */
06765          LOG(1, "Write station data: %s", filename);
06766
06767          /* Create new file... */
06768          if (!(out = fopen(filename, "w")))
06769              ERRMSG("Cannot create file!");
06770
06771          /* Write header... */
06772          fprintf(out,
06773              "# $1 = time [s]\n"
06774              "# $2 = altitude [km]\n"
06775              "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
06776          for (int iq = 0; iq < ctl->nq; iq++)
06777              fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
06778                  ctl->qnt_name[iq], ctl->qnt_unit[iq]);
06779          fprintf(out, "\n");
06780
06781          /* Set geolocation and search radius... */
06782          geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
06783          rmax2 = SQR(ctl->stat_r);
06784      }

```

```

06785
06786  /* Set time interval for output... */
06787  double t0 = t - 0.5 * ctl->dt_mod;
06788  double t1 = t + 0.5 * ctl->dt_mod;
06789
06790  /* Loop over air parcels... */
06791  for (int ip = 0; ip < atm->np; ip++) {
06792
06793      /* Check time... */
06794      if (atm->time[ip] < t0 || atm->time[ip] > t1)
06795          continue;
06796
06797      /* Check time range for station output... */
06798      if (atm->time[ip] < ctl->stat_t0 || atm->time[ip] > ctl->stat_t1)
06799          continue;
06800
06801      /* Check station flag... */
06802      if (ctl->qnt_stat >= 0)
06803          if (atm->q[ctl->qnt_stat][ip])
06804              continue;
06805
06806      /* Get Cartesian coordinates... */
06807      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
06808
06809      /* Check horizontal distance... */
06810      if (DIST2(x0, x1) > rmax2)
06811          continue;
06812
06813      /* Set station flag... */
06814      if (ctl->qnt_stat >= 0)
06815          atm->q[ctl->qnt_stat][ip] = 1;
06816
06817      /* Write data... */
06818      fprintf(out, "%.2f %g %g %g",
06819              atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
06820      for (int iq = 0; iq < ctl->nq; iq++) {
06821          fprintf(out, " ");
06822          fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
06823      }
06824      fprintf(out, "\n");
06825  }
06826
06827  /* Close file... */
06828  if (t == ctl->t_stop)
06829      fclose(out);
06830  }

```

Here is the call graph for this function:



5.24 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) 2013-2023 Forschungszentrum Juelich GmbH

```

```
00018 */
00019
00037 #ifndef LIBTRAC_H
00038 #define LIBTRAC_H
00039
00040 /* -----
00041     Includes...
00042     ----- */
00043
00044 #include <ctype.h>
00045 #include <gsl/gsl_fft_complex.h>
00046 #include <gsl/gsl_math.h>
00047 #include <gsl/gsl_randist.h>
00048 #include <gsl/gsl_rng.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 #ifdef ZFP
00070 #include "zfp.h"
00071 #endif
00072
00073 #ifdef ZSTD
00074 #include "zstd.h"
00075 #endif
00076
00077 /* -----
00078     Constants...
00079     ----- */
00080
00082 #ifndef CPD
00083 #define CPD 1003.5
00084 #endif
00085
00087 #ifndef EPS
00088 #define EPS (MH2O / MA)
00089 #endif
00090
00092 #ifndef G0
00093 #define G0 9.80665
00094 #endif
00095
00097 #ifndef H0
00098 #define H0 7.0
00099 #endif
00100
00102 #ifndef LV
00103 #define LV 2501000.
00104 #endif
00105
00107 #ifndef KB
00108 #define KB 1.3806504e-23
00109 #endif
00110
00112 #ifndef MA
00113 #define MA 28.9644
00114 #endif
00115
00117 #ifndef MH2O
00118 #define MH2O 18.01528
00119 #endif
00120
00122 #ifndef MO3
00123 #define MO3 48.00
00124 #endif
00125
00127 #ifndef P0
00128 #define P0 1013.25
00129 #endif
00130
00132 #ifndef RA
```

```

00133 #define RA (1e3 * RI / MA)
00134 #endif
00135
00137 #ifndef RE
00138 #define RE 6367.421
00139 #endif
00140
00142 #ifndef RI
00143 #define RI 8.3144598
00144 #endif
00145
00147 #ifndef T0
00148 #define T0 273.15
00149 #endif
00150
00151 /* -----
00152     Dimensions...
00153     ----- */
00154
00156 #ifndef LEN
00157 #define LEN 5000
00158 #endif
00159
00161 #ifndef NP
00162 #define NP 10000000
00163 #endif
00164
00166 #ifndef NQ
00167 #define NQ 15
00168 #endif
00169
00171 #ifndef NCSI
00172 #define NCSI 1000000
00173 #endif
00174
00176 #ifndef EP
00177 #define EP 140
00178 #endif
00179
00181 #ifndef EX
00182 #define EX 1201
00183 #endif
00184
00186 #ifndef EY
00187 #define EY 601
00188 #endif
00189
00191 #ifndef NENS
00192 #define NENS 2000
00193 #endif
00194
00196 #ifndef NOBS
00197 #define NOBS 10000000
00198 #endif
00199
00201 #ifndef NTHREADS
00202 #define NTHREADS 512
00203 #endif
00204
00206 #ifndef CY
00207 #define CY 250
00208 #endif
00209
00211 #ifndef CP
00212 #define CP 60
00213 #endif
00214
00216 #ifndef CT
00217 #define CT 12
00218 #endif
00219
00220 /* -----
00221     Macros...
00222     ----- */
00223
00225 #ifdef _OPENACC
00226 #define ALLOC(ptr, type, n) \
00227     if(acc_get_num_devices(acc_device_nvidia) <= 0) \
00228         ERRMSG("Not running on a GPU device!"); \
00229     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00230         ERRMSG("Out of memory!");
00231 #else
00232 #define ALLOC(ptr, type, n) \
00233     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00234         ERRMSG("Out of memory!");
00235 #endif
00236

```

```

00238 #define ARRAY_2D(ix, iy, ny)          \
00239     ((ix) * (ny) + (iy))
00240
00242 #define ARRAY_3D(ix, iy, ny, iz, nz)    \
00243     ((ix)*(ny) + (iy)) * (nz) + (iz)
00244
00246 #define DEG2DX(dlon, lat)               \
00247     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00248
00250 #define DEG2DY(dlat)                   \
00251     ((dlat) * M_PI * RE / 180.)
00252
00254 #define DP2DZ(dp, p)                   \
00255     (- (dp) * H0 / (p))
00256
00258 #define DX2DEG(dx, lat)                \
00259     ((lat) < -89.999 || (lat) > 89.999) ? 0
00260     : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI))
00261
00263 #define DY2DEG(dy)                     \
00264     ((dy) * 180. / (M_PI * RE))
00265
00267 #define DZ2DP(dz, p)                   \
00268     (- (dz) * (p) / H0)
00269
00271 #define DIST(a, b) \
00272     sqrt(DIST2(a, b))
00273
00275 #define DIST2(a, b) \
00276     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00277
00279 #define DOTP(a, b) \
00280     (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00281
00283 #define FMOD(x, y) \
00284     ((x) - (int) ((x) / (y)) * (y))
00285
00287 #define FREAD(ptr, type, size, out) {
00288     if(fread(ptr, sizeof(type), size, out)!=size)
00289         ERRMSG("Error while reading!");
00290 }
00291
00293 #define FWRITE(ptr, type, size, out) {
00294     if(fwrite(ptr, sizeof(type), size, out)!=size)
00295         ERRMSG("Error while writing!");
00296 }
00297
00299 #define INTPOL_INIT \
00300     double cw[3] = {0.0, 0.0, 0.0}; int ci[3] = {0, 0, 0};
00301
00303 #define INTPOL_2D(var, init) \
00304     intpol_met_time_2d(met0, met0->var, met1, met1->var,
00305         atm->time[ip], atm->lon[ip], atm->lat[ip],
00306         &var, ci, cw, init);
00307
00309 #define INTPOL_3D(var, init) \
00310     intpol_met_time_3d(met0, met0->var, met1, met1->var,
00311         atm->time[ip], atm->p[ip],
00312         atm->lon[ip], atm->lat[ip],
00313         &var, ci, cw, init);
00314
00316 #define INTPOL_SPACE_ALL(p, lon, lat) {
00317     intpol_met_space_3d(met, met->z, p, lon, lat, &z, ci, cw, 1);
00318     intpol_met_space_3d(met, met->t, p, lon, lat, &t, ci, cw, 0);
00319     intpol_met_space_3d(met, met->u, p, lon, lat, &u, ci, cw, 0);
00320     intpol_met_space_3d(met, met->v, p, lon, lat, &v, ci, cw, 0);
00321     intpol_met_space_3d(met, met->w, p, lon, lat, &w, ci, cw, 0);
00322     intpol_met_space_3d(met, met->pv, p, lon, lat, &pv, ci, cw, 0);
00323     intpol_met_space_3d(met, met->h2o, p, lon, lat, &h2o, ci, cw, 0);
00324     intpol_met_space_3d(met, met->o3, p, lon, lat, &o3, ci, cw, 0);
00325     intpol_met_space_3d(met, met->lwc, p, lon, lat, &lwc, ci, cw, 0);
00326     intpol_met_space_3d(met, met->iwc, p, lon, lat, &iwc, ci, cw, 0);
00327     intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
00328     intpol_met_space_2d(met, met->ts, lon, lat, &ts, ci, cw, 0);
00329     intpol_met_space_2d(met, met->zs, lon, lat, &zs, ci, cw, 0);
00330     intpol_met_space_2d(met, met->us, lon, lat, &us, ci, cw, 0);
00331     intpol_met_space_2d(met, met->vs, lon, lat, &vs, ci, cw, 0);
00332     intpol_met_space_2d(met, met->pbl, lon, lat, &pbl, ci, cw, 0);
00333     intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
00334     intpol_met_space_2d(met, met->tt, lon, lat, &tt, ci, cw, 0);
00335     intpol_met_space_2d(met, met->zt, lon, lat, &zt, ci, cw, 0);
00336     intpol_met_space_2d(met, met->h2ot, lon, lat, &h2ot, ci, cw, 0);
00337     intpol_met_space_2d(met, met->pct, lon, lat, &pct, ci, cw, 0);
00338     intpol_met_space_2d(met, met->pcb, lon, lat, &pcb, ci, cw, 0);
00339     intpol_met_space_2d(met, met->c1, lon, lat, &c1, ci, cw, 0);
00340     intpol_met_space_2d(met, met->plcl, lon, lat, &plcl, ci, cw, 0);
00341     intpol_met_space_2d(met, met->plfc, lon, lat, &plfc, ci, cw, 0);

```

```

00342 intpol_met_space_2d(met, met->pel, lon, lat, &pel, ci, cw, 0); \
00343 intpol_met_space_2d(met, met->cape, lon, lat, &cape, ci, cw, 0); \
00344 intpol_met_space_2d(met, met->cin, lon, lat, &cin, ci, cw, 0); \
00345 }
00346
00347 #define INTPOL_TIME_ALL(time, p, lon, lat) { \
00348     intpol_met_time_3d(met0, met0->z, met1, met1->z, time, p, lon, lat, &z, ci, cw, 1); \
00349     intpol_met_time_3d(met0, met0->t, met1, met1->t, time, p, lon, lat, &t, ci, cw, 0); \
00350     intpol_met_time_3d(met0, met0->u, met1, met1->u, time, p, lon, lat, &u, ci, cw, 0); \
00351     intpol_met_time_3d(met0, met0->v, met1, met1->v, time, p, lon, lat, &v, ci, cw, 0); \
00352     intpol_met_time_3d(met0, met0->w, met1, met1->w, time, p, lon, lat, &w, ci, cw, 0); \
00353     intpol_met_time_3d(met0, met0->pv, met1, met1->pv, time, p, lon, lat, &pv, ci, cw, 0); \
00354     intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, time, p, lon, lat, &h2o, ci, cw, 0); \
00355     intpol_met_time_3d(met0, met0->o3, met1, met1->o3, time, p, lon, lat, &o3, ci, cw, 0); \
00356     intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, time, p, lon, lat, &lwc, ci, cw, 0); \
00357     intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, time, p, lon, lat, &iwc, ci, cw, 0); \
00358     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, time, lon, lat, &ps, ci, cw, 0); \
00359     intpol_met_time_2d(met0, met0->ts, met1, met1->ts, time, lon, lat, &ts, ci, cw, 0); \
00360     intpol_met_time_2d(met0, met0->zs, met1, met1->zs, time, lon, lat, &zs, ci, cw, 0); \
00361     intpol_met_time_2d(met0, met0->us, met1, met1->us, time, lon, lat, &us, ci, cw, 0); \
00362     intpol_met_time_2d(met0, met0->vs, met1, met1->vs, time, lon, lat, &vs, ci, cw, 0); \
00363     intpol_met_time_2d(met0, met0->pbl, met1, met1->pbl, time, lon, lat, &pbl, ci, cw, 0); \
00364     intpol_met_time_2d(met0, met0->pt, met1, met1->pt, time, lon, lat, &pt, ci, cw, 0); \
00365     intpol_met_time_2d(met0, met0->tt, met1, met1->tt, time, lon, lat, &tt, ci, cw, 0); \
00366     intpol_met_time_2d(met0, met0->zt, met1, met1->zt, time, lon, lat, &zt, ci, cw, 0); \
00367     intpol_met_time_2d(met0, met0->h2ot, met1, met1->h2ot, time, lon, lat, &h2ot, ci, cw, 0); \
00368     intpol_met_time_2d(met0, met0->pct, met1, met1->pct, time, lon, lat, &pct, ci, cw, 0); \
00369     intpol_met_time_2d(met0, met0->pcb, met1, met1->pcb, time, lon, lat, &pcb, ci, cw, 0); \
00370     intpol_met_time_2d(met0, met0->cl, met1, met1->cl, time, lon, lat, &cl, ci, cw, 0); \
00371     intpol_met_time_2d(met0, met0->plcl, met1, met1->plcl, time, lon, lat, &plcl, ci, cw, 0); \
00372     intpol_met_time_2d(met0, met0->plfc, met1, met1->plfc, time, lon, lat, &plfc, ci, cw, 0); \
00373     intpol_met_time_2d(met0, met0->pel, met1, met1->pel, time, lon, lat, &pel, ci, cw, 0); \
00374     intpol_met_time_2d(met0, met0->cape, met1, met1->cape, time, lon, lat, &cape, ci, cw, 0); \
00375     intpol_met_time_2d(met0, met0->cin, met1, met1->cin, time, lon, lat, &cin, ci, cw, 0); \
00376 }
00377
00378 #define LAPSE(p1, t1, p2, t2) \
00379     (1e3 * G0 / RA * ((t2) - (t1)) / ((t2) + (t1)) \
00380     * ((p2) + (p1)) / ((p2) - (p1))) \
00381
00382 #define LIN(x0, y0, x1, y1, x) \
00383     ((y0) + ((y1) - (y0)) / ((x1) - (x0)) * ((x) - (x0))) \
00384
00385 #define NC(cmd) { \
00386     int nc_result=(cmd); \
00387     if(nc_result!=NC_NOERR) \
00388         ERRMSG("%s", nc_strerror(nc_result)); \
00389 }
00390
00391 #define NC_DEF_VAR(varname, type, ndims, dims, long_name, units) { \
00392     NC(nc_def_var(ncid, varname, type, ndims, dims, &varid)); \
00393     NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name)); \
00394     NC(nc_put_att_text(ncid, varid, "units", strlen(units), units)); \
00395 }
00396
00397 #define NC_GET_DOUBLE(varname, ptr, force) { \
00398     if(force) { \
00399         NC(nc_inq_varid(ncid, varname, &varid)); \
00400         NC(nc_get_var_double(ncid, varid, ptr)); \
00401     } else { \
00402         if(nc_inq_varid(ncid, varname, &varid) == NC_NOERR) { \
00403             NC(nc_get_var_double(ncid, varid, ptr)); \
00404         } else \
00405             WARN("netCDF variable %s is missing!", varname); \
00406     } \
00407 }
00408
00409 #define NC_INQ_DIM(dimname, ptr, min, max) { \
00410     int dimid; size_t naux; \
00411     NC(nc_inq_dimid(ncid, dimname, &dimid)); \
00412     NC(nc_inq_dimlen(ncid, dimid, &naux)); \
00413     *ptr = (int)naux; \
00414     if ((*ptr) < (min) || (*ptr) > (max)) \
00415         ERRMSG("Dimension %s is out of range!", dimname); \
00416 }
00417
00418 #define NC_PUT_DOUBLE(varname, ptr, hyperslab) { \
00419     NC(nc_inq_varid(ncid, varname, &varid)); \
00420     if(hyperslab) { \
00421         NC(nc_put_vara_double(ncid, varid, start, count, ptr)); \
00422     } else { \
00423         NC(nc_put_var_double(ncid, varid, ptr)); \
00424     } \
00425 }
00426
00427 #define NC_PUT_INT(varname, ptr, hyperslab) { \
00428     NC(nc_inq_varid(ncid, varname, &varid)); \
00429 }

```

```

00438     if(hyperslab) {
00439         NC(nc_put_vara_int(ncid, varid, start, count, ptr));
00440     } else {
00441         NC(nc_put_var_int(ncid, varid, ptr));
00442     }
00443 }
00444
00446 #define NC_PUT_ATT(varname, attname, text) {
00447     NC(nc_inq_varid(ncid, varname, &varid));
00448     NC(nc_put_att_text(ncid, varid, attname, strlen(text), text));
00449 }
00450
00452 #define NC_PUT_ATT_GLOBAL(attname, text)
00453     NC(nc_put_att_text(ncid, NC_GLOBAL, attname, strlen(text), text));
00454
00456 #define NC_PUT_FLOAT(varname, ptr, hyperslab) {
00457     NC(nc_inq_varid(ncid, varname, &varid));
00458     if(hyperslab) {
00459         NC(nc_put_vara_float(ncid, varid, start, count, ptr));
00460     } else {
00461         NC(nc_put_var_float(ncid, varid, ptr));
00462     }
00463 }
00464
00466 #define NN(x0, y0, x1, y1, x) \
00467     (fabs((x) - (x0)) <= fabs((x) - (x1)) ? (y0) : (y1))
00468
00470 #define NORM(a) \
00471     sqrt(DOTP(a, a))
00472
00474 #define P(z) \
00475     (P0 * exp(-(z) / H0))
00476
00478 #define PSAT(t) \
00479     (6.112 * exp(17.62 * ((t) - T0) / (243.12 + (t) - T0)))
00480
00482 #define PSICE(t) \
00483     (6.112 * exp(22.46 * ((t) - T0) / (272.62 + (t) - T0)))
00484
00486 #define PW(p, h2o) \
00487     ((p) * GSL_MAX((h2o), 0.1e-6) \
00488     / (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
00489
00491 #define RH(p, t, h2o) \
00492     (PW(p, h2o) / PSAT(t) * 100.)
00493
00495 #define RHICE(p, t, h2o) \
00496     (PW(p, h2o) / PSICE(t) * 100.)
00497
00499 #define RHO(p, t) \
00500     (100. * (p) / (RA * (t)))
00501
00503 #define SET_ATM(qnt, val) \
00504     if (ctl->qnt >= 0) \
00505         atm->q[ctl->qnt][ip] = val;
00506
00508 #define SET_QNT(qnt, name, longname, unit) \
00509     if (strcasecmp(ctl->qnt_name[iq], name) == 0) { \
00510         ctl->qnt = iq; \
00511         sprintf(ctl->qnt_longname[iq], longname); \
00512         sprintf(ctl->qnt_unit[iq], unit); \
00513     } else
00514
00516 #define SH(h2o) \
00517     (EPS * GSL_MAX((h2o), 0.1e-6))
00518
00520 #define SQR(x) \
00521     ((x)*(x))
00522
00524 #define SWAP(x, y, type) \
00525     do {type tmp = x; x = y; y = tmp;} while(0);
00526
00528 #define TDEW(p, h2o) \
00529     (T0 + 243.12 * log(PW((p), (h2o)) / 6.112) \
00530     / (17.62 - log(PW((p), (h2o)) / 6.112)))
00531
00533 #define TICE(p, h2o) \
00534     (T0 + 272.62 * log(PW((p), (h2o)) / 6.112) \
00535     / (22.46 - log(PW((p), (h2o)) / 6.112)))
00536
00538 #define THETA(p, t) \
00539     ((t) * pow(1000. / (p), 0.286))
00540
00542 #define THETA_VIRT(p, t, h2o) \
00543     (TVIRT(THETA((p), (t)), GSL_MAX((h2o), 0.1e-6)))
00544
00546 #define TOK(line, tok, format, var) {

```



```

00547     if(((tok)=strtok((line), " \t"))) {
00548         if(sscanf(tok, format, &(var))!=1) continue;
00549     } else ERRMSG("Error while reading!");
00550 }
00551
00553 #define TVIRT(t, h2o) \
00554     ((t) * (1. + (1. - EPS) * GSL_MAX((h2o), 0.1e-6)))
00555
00557 #define Z(p) \
00558     (H0 * log(P0 / (p)))
00559
00561 #define ZDIFF(lnp0, t0, h2o0, lnp1, t1, h2o1) \
00562     (RI / MA / G0 * 0.5 * (TVIRT((t0), (h2o0)) + TVIRT((t1), (h2o1))) \
00563     * ((lnp0) - (lnp1)))
00564
00566 #define ZETA(ps, p, t) \
00567     ((p) / (ps) <= 0.3 ? 1. : \
00568     sin(M_PI / 2. * (1. - (p) / (ps)) / (1. - 0.3))) \
00569     * THETA((p), (t)))
00570
00571 /* -----
00572     Log messages...
00573     ----- */
00574
00576 #ifndef LOGLEV
00577 #define LOGLEV 2
00578 #endif
00579
00581 #define LOG(level, ...) { \
00582     if(level >= 2) \
00583         printf(" "); \
00584     if(level <= LOGLEV) { \
00585         printf(__VA_ARGS__); \
00586         printf("\n"); \
00587     } \
00588 }
00589
00591 #define WARN(...) { \
00592     printf("\nWarning (%s, %s, %d): ", __FILE__, __func__, __LINE__); \
00593     LOG(0, __VA_ARGS__); \
00594 }
00595
00597 #define ERRMSG(...) { \
00598     printf("\nError (%s, %s, %d): ", __FILE__, __func__, __LINE__); \
00599     LOG(0, __VA_ARGS__); \
00600     exit(EXIT_FAILURE); \
00601 }
00602
00604 #define PRINT(format, var) \
00605     printf("Print (%s, %s, %d): %s= "format"\n", \
00606     __FILE__, __func__, __LINE__, #var, var);
00607
00608 /* -----
00609     Timers...
00610     ----- */
00611
00613 #define NTIMER 100
00614
00616 #define PRINT_TIMERS \
00617     timer("END", "END", 1);
00618
00620 #define SELECT_TIMER(id, group, color) { \
00621     NVTX_POP; \
00622     NVTX_PUSH(id, color); \
00623     timer(id, group, 0); \
00624 }
00625
00627 #define START_TIMERS \
00628     NVTX_PUSH("START", NVTX_CPU);
00629
00631 #define STOP_TIMERS \
00632     NVTX_POP;
00633
00634 /* -----
00635     NVIDIA Tools Extension (NVTX)...
00636     ----- */
00637
00638 #ifdef NVTX
00639 #include "nvToolsExt.h"
00640
00642 #define NVTX_CPU 0xFFADD8E6
00643
00645 #define NVTX_GPU 0xFF00008B
00646
00648 #define NVTX_H2D 0xFFFFFFFF00
00649
00651 #define NVTX_D2H 0xFFFF8800

```

```

00652
00654 #define NVTX_READ 0xFFFFCCCB
00655
00657 #define NVTX_WRITE 0xFF8B0000
00658
00660 #define NVTX_PUSH(range_title, range_color) {           \
00661     nvtxEventAttributes_t eventAttrib = {0};           \
00662     eventAttrib.version = NVTX_VERSION;                 \
00663     eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;   \
00664     eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII; \
00665     eventAttrib.colorType = NVTX_COLOR_ARGB;           \
00666     eventAttrib.color = range_color;                   \
00667     eventAttrib.message.ascii = range_title;           \
00668     nvtxRangePushEx(&eventAttrib);                     \
00669 }
00670
00672 #define NVTX_POP {                                     \
00673     nvtxRangePop();                                   \
00674 }
00675 #else
00676
00677 /* Empty definitions of NVTX_PUSH and NVTX_POP... */
00678 #define NVTX_PUSH(range_title, range_color) {}
00679 #define NVTX_POP {}
00680 #endif
00681
00682 /* -----
00683 Thrust...
00684 ----- */
00685
00687 void thrustSortWrapper(
00688     double *__restrict__ c,
00689     int n,
00690     int *__restrict__ index);
00691
00692 /* -----
00693 Structs...
00694 ----- */
00695
00697 typedef struct {
00698     int vert_coord_ap;
00700     int vert_coord_met;
00701
00703     int vert_vel;
00704
00706     int clams_met_data;
00709     int chunkszhint;
00710
00712     int read_mode;
00713
00715     int nq;
00718     char qnt_name[NQ][LEN];
00721     char qnt_longname[NQ][LEN];
00724     char qnt_unit[NQ][LEN];
00727     char qnt_format[NQ][LEN];
00730     int qnt_idx;
00733     int qnt_ens;
00736     int qnt_stat;
00739     int qnt_m;
00742     int qnt_vmr;
00745     int qnt_rp;
00748     int qnt_rhop;
00751     int qnt_ps;
00754     int qnt_ts;
00757     int qnt_zs;
00760     int qnt_us;
00763     int qnt_vs;
00766
00767

```

```
00769     int qnt_pbl;
00770
00772     int qnt_pt;
00773
00775     int qnt_tt;
00776
00778     int qnt_zt;
00779
00781     int qnt_h2ot;
00782
00784     int qnt_z;
00785
00787     int qnt_p;
00788
00790     int qnt_t;
00791
00793     int qnt_rho;
00794
00796     int qnt_u;
00797
00799     int qnt_v;
00800
00802     int qnt_w;
00803
00805     int qnt_h2o;
00806
00808     int qnt_o3;
00809
00811     int qnt_lwc;
00812
00814     int qnt_iwc;
00815
00817     int qnt_pct;
00818
00820     int qnt_pcb;
00821
00823     int qnt_cl;
00824
00826     int qnt_plcl;
00827
00829     int qnt_plfc;
00830
00832     int qnt_pel;
00833
00835     int qnt_cape;
00836
00838     int qnt_cin;
00839
00841     int qnt_hno3;
00842
00844     int qnt_oh;
00845
00847     int qnt_vmrimpl;
00848
00850     int qnt_mloss_oh;
00851
00853     int qnt_mloss_h2o2;
00854
00856     int qnt_mloss_wet;
00857
00859     int qnt_mloss_dry;
00860
00862     int qnt_mloss_decay;
00863
00865     int qnt_psat;
00866
00868     int qnt_psice;
00869
00871     int qnt_pw;
00872
00874     int qnt_sh;
00875
00877     int qnt_rh;
00878
00880     int qnt_rhice;
00881
00883     int qnt_theta;
00884
00886     int qnt_zeta;
00887
00889     int qnt_tvirt;
00890
00892     int qnt_lapse;
00893
00895     int qnt_vh;
00896
00898     int qnt_vz;
```

```
00899
00901     int qnt_pv;
00902
00904     int qnt_tdew;
00905
00907     int qnt_tice;
00908
00910     int qnt_tsts;
00911
00913     int qnt_tnat;
00914
00916     int direction;
00917
00919     double t_start;
00920
00922     double t_stop;
00923
00925     double dt_mod;
00926
00928     char metbase[LEN];
00929
00931     double dt_met;
00932
00934     int met_type;
00935
00937     int met_nc_scale;
00938
00940     int met_dx;
00941
00943     int met_dy;
00944
00946     int met_dp;
00947
00949     int met_sx;
00950
00952     int met_sy;
00953
00955     int met_sp;
00956
00958     double met_detrend;
00959
00961     int met_np;
00962
00964     double met_p[EP];
00965
00967     int met_geopot_sx;
00968
00970     int met_geopot_sy;
00971
00973     int met_relhum;
00974
00977     int met_tropo;
00978
00980     double met_tropo_lapse;
00981
00983     int met_tropo_nlev;
00984
00986     double met_tropo_lapse_sep;
00987
00989     int met_tropo_nlev_sep;
00990
00992     double met_tropo_pv;
00993
00995     double met_tropo_theta;
00996
00998     int met_tropo_spline;
00999
01001     int met_cloud;
01002
01004     double met_cloud_min;
01005
01007     double met_dt_out;
01008
01010     int met_cache;
01011
01013     double sort_dt;
01014
01017     int isosurf;
01018
01020     char balloon[LEN];
01021
01023     int advect;
01024
01026     int reflect;
01027
01029     double turb_dx_trop;
01030
```

```
01032 double turb_dx_strat;
01033
01035 double turb_dz_trop;
01036
01038 double turb_dz_strat;
01039
01041 double turb_mesox;
01042
01044 double turb_mesoz;
01045
01047 double conv_cape;
01048
01050 double conv_cin;
01051
01053 double conv_dt;
01054
01056 int conv_mix;
01057
01059 int conv_mix_bot;
01060
01062 int conv_mix_top;
01063
01065 double bound_mass;
01066
01068 double bound_mass_trend;
01069
01071 double bound_vmr;
01072
01074 double bound_vmr_trend;
01075
01077 double bound_lat0;
01078
01080 double bound_lat1;
01081
01083 double bound_p0;
01084
01086 double bound_p1;
01087
01089 double bound_dps;
01090
01092 double bound_dzs;
01093
01095 double bound_zetas;
01096
01098 int bound_pbl;
01099
01101 char species[LEN];
01102
01104 double molmass;
01105
01107 double tdec_trop;
01108
01110 double tdec_strat;
01111
01113 char clim_oh_filename[LEN];
01114
01116 char clim_h2o2_filename[LEN];
01117
01119 int oh_chem_reaction;
01120
01122 double oh_chem[4];
01123
01125 double oh_chem_beta;
01126
01128 double h2o2_chem_cc;
01129
01131 int h2o2_chem_reaction;
01132
01134 int chemgrid_nz;
01135
01137 double chemgrid_z0;
01138
01140 double chemgrid_z1;
01141
01143 int chemgrid_nx;
01144
01146 double chemgrid_lon0;
01147
01149 double chemgrid_lon1;
01150
01152 int chemgrid_ny;
01153
01155 double chemgrid_lat0;
01156
01158 double chemgrid_lat1;
01159
01161 double dry_depo_dp;
```

```
01162
01164 double dry_depo_vdep;
01165
01167 double wet_depo_pre[2];
01168
01170 double wet_depo_bc_a;
01171
01173 double wet_depo_bc_b;
01174
01176 double wet_depo_ic_a;
01177
01179 double wet_depo_ic_b;
01180
01182 double wet_depo_ic_h[3];
01183
01185 double wet_depo_bc_h[2];
01186
01188 double wet_depo_ic_ret_ratio;
01189
01191 double wet_depo_bc_ret_ratio;
01192
01194 double psc_h2o;
01195
01197 double psc_hno3;
01198
01200 char atm_basename[LEN];
01201
01203 char atm_gpfile[LEN];
01204
01206 double atm_dt_out;
01207
01209 int atm_filter;
01210
01212 int atm_stride;
01213
01215 int atm_type;
01216
01218 char csi_basename[LEN];
01219
01221 double csi_dt_out;
01222
01224 char csi_obsfile[LEN];
01225
01227 double csi_obsmin;
01228
01230 double csi_modmin;
01231
01233 int csi_nz;
01234
01236 double csi_z0;
01237
01239 double csi_z1;
01240
01242 int csi_nx;
01243
01245 double csi_lon0;
01246
01248 double csi_lon1;
01249
01251 int csi_ny;
01252
01254 double csi_lat0;
01255
01257 double csi_lat1;
01258
01260 char ens_basename[LEN];
01261
01263 double ens_dt_out;
01264
01266 char grid_basename[LEN];
01267
01269 char grid_gpfile[LEN];
01270
01272 double grid_dt_out;
01273
01275 int grid_sparse;
01276
01278 int grid_nz;
01279
01281 double grid_z0;
01282
01284 double grid_z1;
01285
01287 int grid_nx;
01288
01290 double grid_lon0;
01291
```

```
01293 double grid_lon1;
01294
01296 int grid_ny;
01297
01299 double grid_lat0;
01300
01302 double grid_lat1;
01303
01305 int grid_type;
01306
01308 char prof_basename[LEN];
01309
01311 char prof_obsfile[LEN];
01312
01314 int prof_nz;
01315
01317 double prof_z0;
01318
01320 double prof_z1;
01321
01323 int prof_nx;
01324
01326 double prof_lon0;
01327
01329 double prof_lon1;
01330
01332 int prof_ny;
01333
01335 double prof_lat0;
01336
01338 double prof_lat1;
01339
01341 char sample_basename[LEN];
01342
01344 char sample_obsfile[LEN];
01345
01347 double sample_dx;
01348
01350 double sample_dz;
01351
01353 char stat_basename[LEN];
01354
01356 double stat_lon;
01357
01359 double stat_lat;
01360
01362 double stat_r;
01363
01365 double stat_t0;
01366
01368 double stat_t1;
01369
01370 } ctl_t;
01371
01373 typedef struct {
01374
01376 int np;
01377
01379 double time[NP];
01380
01382 double p[NP];
01383
01385 double zeta[NP];
01386
01388 double lon[NP];
01389
01391 double lat[NP];
01392
01394 double q[NQ][NP];
01395
01396 } atm_t;
01397
01399 typedef struct {
01400
01402 double iso_var[NP];
01403
01405 double iso_ps[NP];
01406
01408 double iso_ts[NP];
01409
01411 int iso_n;
01412
01414 float uvwp[NP][3];
01415
01416 } cache_t;
01417
01419 typedef struct {
```

```
01420
01422     int tropo_ntime;
01423
01425     int tropo_nlat;
01426
01428     double tropo_time[12];
01429
01431     double tropo_lat[73];
01432
01434     double tropo[12][73];
01435
01437     int hno3_ntime;
01438
01440     int hno3_nlat;
01441
01443     int hno3_np;
01444
01446     double hno3_time[12];
01447
01449     double hno3_lat[18];
01450
01452     double hno3_p[10];
01453
01455     double hno3[12][18][10];
01456
01458     int oh_ntime;
01459
01461     int oh_nlat;
01462
01464     int oh_np;
01465
01467     double oh_time[CT];
01468
01470     double oh_lat[CY];
01471
01473     double oh_p[CP];
01474
01476     double oh[CT][CP][CY];
01477
01479     int h2o2_ntime;
01480
01482     int h2o2_nlat;
01483
01485     int h2o2_np;
01486
01488     double h2o2_time[CT];
01489
01491     double h2o2_lat[CY];
01492
01494     double h2o2_p[CP];
01495
01497     double h2o2[CT][CP][CY];
01498
01499 } clim_t;
01500
01502 typedef struct {
01503
01505     double time;
01506
01508     int nx;
01509
01511     int ny;
01512
01514     int np;
01515
01517     double lon[EX];
01518
01520     double lat[EY];
01521
01523     double p[EP];
01524
01526     float ps[EX][EY];
01527
01529     float ts[EX][EY];
01530
01532     float zs[EX][EY];
01533
01535     float us[EX][EY];
01536
01538     float vs[EX][EY];
01539
01541     float pbl[EX][EY];
01542
01544     float pt[EX][EY];
01545
01547     float tt[EX][EY];
01548
```



```

01550 float zt[EX][EY];
01551
01553 float h2ot[EX][EY];
01554
01556 float pct[EX][EY];
01557
01559 float pcb[EX][EY];
01560
01562 float cl[EX][EY];
01563
01565 float plcl[EX][EY];
01566
01568 float plfc[EX][EY];
01569
01571 float pel[EX][EY];
01572
01574 float cape[EX][EY];
01575
01577 float cin[EX][EY];
01578
01580 float z[EX][EY][EP];
01581
01583 float t[EX][EY][EP];
01584
01586 float u[EX][EY][EP];
01587
01589 float v[EX][EY][EP];
01590
01592 float w[EX][EY][EP];
01593
01595 float pv[EX][EY][EP];
01596
01598 float h2o[EX][EY][EP];
01599
01601 float o3[EX][EY][EP];
01602
01604 float lwc[EX][EY][EP];
01605
01607 float iwc[EX][EY][EP];
01608
01610 float pl[EX][EY][EP];
01611
01613 float patp[EX][EY][EP];
01614
01616 float zeta[EX][EY][EP];
01617
01619 float zeta_dot[EX][EY][EP];
01620
01621 #ifdef UVW
01623 float uvw[EX][EY][EP][3];
01624 #endif
01625
01626 } met_t;
01627
01628 /* -----
01629 Functions...
01630 ----- */
01631
01633 double buoyancy_frequency(
01634     double p0,
01635     double t0,
01636     double p1,
01637     double t1);
01638
01640 void cart2geo(
01641     double *x,
01642     double *z,
01643     double *lon,
01644     double *lat);
01645
01647 #ifdef _OPENACC
01648 #pragma acc routine (check_finite)
01649 #endif
01650 int check_finite(
01651     const double x);
01652
01654 #ifdef _OPENACC
01655 #pragma acc routine (clim_hno3)
01656 #endif
01657 double clim_hno3(
01658     clim_t * clim,
01659     double t,
01660     double lat,
01661     double p);
01662
01664 void clim_hno3_init(
01665     clim_t * clim);

```

```
01666
01668 #ifdef _OPENACC
01669 #pragma acc routine (clim_oh)
01670 #endif
01671 double clim_oh(
01672     clim_t * clim,
01673     double t,
01674     double lat,
01675     double p);
01676
01678 #ifdef _OPENACC
01679 #pragma acc routine (clim_oh_diurnal)
01680 #endif
01681 double clim_oh_diurnal(
01682     ctl_t * ctl,
01683     clim_t * clim,
01684     double t,
01685     double p,
01686     double lon,
01687     double lat);
01688
01690 void clim_oh_init(
01691     ctl_t * ctl,
01692     clim_t * clim);
01693
01695 double clim_oh_init_help(
01696     double beta,
01697     double time,
01698     double lat);
01699
01701 #ifdef _OPENACC
01702 #pragma acc routine (clim_h2o2)
01703 #endif
01704 double clim_h2o2(
01705     clim_t * clim,
01706     double t,
01707     double lat,
01708     double p);
01709
01711 void clim_h2o2_init(
01712     ctl_t * ctl,
01713     clim_t * clim);
01714
01716 #ifdef _OPENACC
01717 #pragma acc routine (clim_tropo)
01718 #endif
01719 double clim_tropo(
01720     clim_t * clim,
01721     double t,
01722     double lat);
01723
01725 void clim_tropo_init(
01726     clim_t * clim);
01727
01729 void compress_pack(
01730     char *varname,
01731     float *array,
01732     size_t nxy,
01733     size_t nz,
01734     int decompress,
01735     FILE * inout);
01736
01738 #ifdef ZFP
01739 void compress_zfp(
01740     char *varname,
01741     float *array,
01742     int nx,
01743     int ny,
01744     int nz,
01745     int precision,
01746     double tolerance,
01747     int decompress,
01748     FILE * inout);
01749 #endif
01750
01752 #ifdef ZSTD
01753 void compress_zstd(
01754     char *varname,
01755     float *array,
01756     size_t n,
01757     int decompress,
01758     FILE * inout);
01759 #endif
01760
01762 void day2doy(
01763     int year,
01764     int mon,
```

```
01765     int day,
01766     int *doy);
01767
01768 void doy2day(
01769     int year,
01770     int doy,
01771     int *mon,
01772     int *day);
01773
01774 void geo2cart(
01775     double z,
01776     double lon,
01777     double lat,
01778     double *x);
01779
01780 void get_met(
01781     ctl_t * ctl,
01782     clim_t * clim,
01783     double t,
01784     met_t ** met0,
01785     met_t ** met1);
01786
01787 void get_met_help(
01788     ctl_t * ctl,
01789     double t,
01790     int direct,
01791     char *metbase,
01792     double dt_met,
01793     char *filename);
01794
01795 void get_met_replace(
01796     char *orig,
01797     char *search,
01798     char *repl);
01799
01800 #ifdef _OPENACC
01801 #pragma acc routine (intpol_met_space_3d)
01802 #endif
01803 void intpol_met_space_3d(
01804     met_t * met,
01805     float array[EX][EY][EP],
01806     double p,
01807     double lon,
01808     double lat,
01809     double *var,
01810     int *ci,
01811     double *cw,
01812     int init);
01813
01814 #ifdef _OPENACC
01815 #pragma acc routine (intpol_met_space_2d)
01816 #endif
01817 void intpol_met_space_2d(
01818     met_t * met,
01819     float array[EX][EY],
01820     double lon,
01821     double lat,
01822     double *var,
01823     int *ci,
01824     double *cw,
01825     int init);
01826
01827 #ifdef UVW
01828 #ifdef _OPENACC
01829 #pragma acc routine (intpol_met_space_uvw)
01830 #endif
01831 void intpol_met_space_uvw(
01832     met_t * met,
01833     double p,
01834     double lon,
01835     double lat,
01836     double *u,
01837     double *v,
01838     double *w,
01839     int *ci,
01840     double *cw,
01841     int init);
01842 #endif
01843
01844 #ifdef _OPENACC
01845 #pragma acc routine (intpol_met_time_3d)
01846 #endif
01847 void intpol_met_time_3d(
01848     met_t * met0,
01849     float array0[EX][EY][EP],
01850     met_t * met1,
01851     float array1[EX][EY][EP],
```

```
01861 double ts,
01862 double p,
01863 double lon,
01864 double lat,
01865 double *var,
01866 int *ci,
01867 double *cw,
01868 int init);
01869
01871 #ifdef _OPENACC
01872 #pragma acc routine (intpol_met_time_2d)
01873 #endif
01874 void intpol_met_time_2d(
01875     met_t * met0,
01876     float array0[EX][EY],
01877     met_t * met1,
01878     float array1[EX][EY],
01879     double ts,
01880     double lon,
01881     double lat,
01882     double *var,
01883     int *ci,
01884     double *cw,
01885     int init);
01886
01888 #ifdef UVW
01889 #ifdef _OPENACC
01890 #pragma acc routine (intpol_met_time_uvw)
01891 #endif
01892 void intpol_met_time_uvw(
01893     met_t * met0,
01894     met_t * met1,
01895     double ts,
01896     double p,
01897     double lon,
01898     double lat,
01899     double *u,
01900     double *v,
01901     double *w);
01902 #endif
01903
01905 void jsec2time(
01906     double jsec,
01907     int *year,
01908     int *mon,
01909     int *day,
01910     int *hour,
01911     int *min,
01912     int *sec,
01913     double *remain);
01914
01916 #ifdef _OPENACC
01917 #pragma acc routine (lapse_rate)
01918 #endif
01919 double lapse_rate(
01920     double t,
01921     double h2o);
01922
01924 #ifdef _OPENACC
01925 #pragma acc routine (locate_irr)
01926 #endif
01927 int locate_irr(
01928     double *xx,
01929     int n,
01930     double x);
01931
01933 #ifdef _OPENACC
01934 #pragma acc routine (locate_reg)
01935 #endif
01936 int locate_reg(
01937     double *xx,
01938     int n,
01939     double x);
01940
01942 #ifdef _OPENACC
01943 #pragma acc routine (nat_temperature)
01944 #endif
01945 double nat_temperature(
01946     double p,
01947     double h2o,
01948     double hno3);
01949
01951 void quicksort(
01952     double arr[],
01953     int brr[],
01954     int low,
01955     int high);
```

```
01956
01958 int quicksort_partition(
01959     double arr[],
01960     int brr[],
01961     int low,
01962     int high);
01963
01965 int read_atm(
01966     const char *filename,
01967     ctl_t * ctl,
01968     atm_t * atm);
01969
01971 int read_atm_asc(
01972     const char *filename,
01973     ctl_t * ctl,
01974     atm_t * atm);
01975
01977 int read_atm_bin(
01978     const char *filename,
01979     ctl_t * ctl,
01980     atm_t * atm);
01981
01983 int read_atm_clams(
01984     const char *filename,
01985     ctl_t * ctl,
01986     atm_t * atm);
01987
01989 int read_atm_nc(
01990     const char *filename,
01991     ctl_t * ctl,
01992     atm_t * atm);
01993
01995 void read_clim(
01996     ctl_t * ctl,
01997     clim_t * clim);
01998
02000 void read_ctl(
02001     const char *filename,
02002     int argc,
02003     char *argv[],
02004     ctl_t * ctl);
02005
02007 int read_met(
02008     char *filename,
02009     ctl_t * ctl,
02010     clim_t * clim,
02011     met_t * met);
02012
02014 void read_met_bin_2d(
02015     FILE * out,
02016     met_t * met,
02017     float var[EX][EY],
02018     char *varname);
02019
02021 void read_met_bin_3d(
02022     FILE * in,
02023     ctl_t * ctl,
02024     met_t * met,
02025     float var[EX][EY][EP],
02026     char *varname,
02027     int precision,
02028     double tolerance);
02029
02031 void read_met_cape(
02032     clim_t * clim,
02033     met_t * met);
02034
02036 void read_met_cloud(
02037     ctl_t * ctl,
02038     met_t * met);
02039
02041 void read_met_detrend(
02042     ctl_t * ctl,
02043     met_t * met);
02044
02046 void read_met_extrapolate(
02047     met_t * met);
02048
02050 void read_met_geopot(
02051     ctl_t * ctl,
02052     met_t * met);
02053
02055 void read_met_grid(
02056     char *filename,
02057     int ncid,
02058     ctl_t * ctl,
02059     met_t * met);
```

```
02060
02062 void read_met_levels(
02063     int ncid,
02064     ctl_t * ctl,
02065     met_t * met);
02066
02068 void read_met_ml2pl(
02069     ctl_t * ctl,
02070     met_t * met,
02071     float var[EX][EY][EP]);
02072
02074 int read_met_nc_2d(
02075     int ncid,
02076     char *varname,
02077     char *varname2,
02078     ctl_t * ctl,
02079     met_t * met,
02080     float dest[EX][EY],
02081     float scl,
02082     int init);
02083
02085 int read_met_nc_3d(
02086     int ncid,
02087     char *varname,
02088     char *varname2,
02089     ctl_t * ctl,
02090     met_t * met,
02091     float dest[EX][EY][EP],
02092     float scl,
02093     int init);
02094
02096 void read_met_pbl(
02097     met_t * met);
02098
02100 void read_met_periodic(
02101     met_t * met);
02102
02104 void read_met_pv(
02105     met_t * met);
02106
02108 void read_met_sample(
02109     ctl_t * ctl,
02110     met_t * met);
02111
02113 void read_met_surface(
02114     int ncid,
02115     met_t * met,
02116     ctl_t * ctl);
02117
02119 void read_met_tropo(
02120     ctl_t * ctl,
02121     clim_t * clim,
02122     met_t * met);
02123
02125 void read_obs(
02126     char *filename,
02127     double *rt,
02128     double *rz,
02129     double *rlon,
02130     double *rlat,
02131     double *robs,
02132     int *nobs);
02133
02135 double scan_ctl(
02136     const char *filename,
02137     int argc,
02138     char *argv[],
02139     const char *varname,
02140     int arridx,
02141     const char *defvalue,
02142     char *value);
02143
02145 #ifdef _OPENACC
02146 #pragma acc routine (sedi)
02147 #endif
02148 double sedi(
02149     double p,
02150     double T,
02151     double rp,
02152     double rhop);
02153
02155 void spline(
02156     double *x,
02157     double *y,
02158     int n,
02159     double *x2,
02160     double *y2,
```

```
02161     int n2,
02162     int method);
02163
02165 #ifdef _OPENACC
02166 #pragma acc routine (stddev)
02167 #endif
02168 float stddev(
02169     float *data,
02170     int n);
02171
02173 #ifdef _OPENACC
02174 #pragma acc routine (sza)
02175 #endif
02176 double sza(
02177     double sec,
02178     double lon,
02179     double lat);
02180
02182 void time2jsec(
02183     int year,
02184     int mon,
02185     int day,
02186     int hour,
02187     int min,
02188     int sec,
02189     double remain,
02190     double *jsec);
02191
02193 void timer(
02194     const char *name,
02195     const char *group,
02196     int output);
02197
02199 #ifdef _OPENACC
02200 #pragma acc routine (tropo_weight)
02201 #endif
02202 double tropo_weight(
02203     clim_t * clim,
02204     double t,
02205     double lat,
02206     double p);
02207
02209 void write_atm(
02210     const char *filename,
02211     ctl_t * ctl,
02212     atm_t * atm,
02213     double t);
02214
02216 void write_atm_asc(
02217     const char *filename,
02218     ctl_t * ctl,
02219     atm_t * atm,
02220     double t);
02221
02223 void write_atm_bin(
02224     const char *filename,
02225     ctl_t * ctl,
02226     atm_t * atm);
02227
02229 void write_atm_clams(
02230     ctl_t * ctl,
02231     atm_t * atm,
02232     double t);
02233
02235 void write_atm_nc(
02236     const char *filename,
02237     ctl_t * ctl,
02238     atm_t * atm);
02239
02241 void write_csi(
02242     const char *filename,
02243     ctl_t * ctl,
02244     atm_t * atm,
02245     double t);
02246
02248 void write_ens(
02249     const char *filename,
02250     ctl_t * ctl,
02251     atm_t * atm,
02252     double t);
02253
02255 void write_grid(
02256     const char *filename,
02257     ctl_t * ctl,
02258     met_t * met0,
02259     met_t * met1,
02260     atm_t * atm,
```

```

02261     double t);
02262
02264 void write_grid_asc(
02265     const char *filename,
02266     ctl_t * ctl,
02267     double *cd,
02268     double *vmr_expl,
02269     double *vmr_impl,
02270     double t,
02271     double *z,
02272     double *lon,
02273     double *lat,
02274     double *area,
02275     double dz,
02276     int *np);
02277
02279 void write_grid_nc(
02280     const char *filename,
02281     ctl_t * ctl,
02282     double *cd,
02283     double *vmr_expl,
02284     double *vmr_impl,
02285     double t,
02286     double *z,
02287     double *lon,
02288     double *lat,
02289     double *area,
02290     double dz,
02291     int *np);
02292
02294 int write_met(
02295     char *filename,
02296     ctl_t * ctl,
02297     met_t * met);
02298
02300 void write_met_bin_2d(
02301     FILE * out,
02302     met_t * met,
02303     float var[EX][EY],
02304     char *varname);
02305
02307 void write_met_bin_3d(
02308     FILE * out,
02309     ctl_t * ctl,
02310     met_t * met,
02311     float var[EX][EY][EP],
02312     char *varname,
02313     int precision,
02314     double tolerance);
02315
02317 void write_prof(
02318     const char *filename,
02319     ctl_t * ctl,
02320     met_t * met0,
02321     met_t * met1,
02322     atm_t * atm,
02323     double t);
02324
02326 void write_sample(
02327     const char *filename,
02328     ctl_t * ctl,
02329     met_t * met0,
02330     met_t * met1,
02331     atm_t * atm,
02332     double t);
02333
02335 void write_station(
02336     const char *filename,
02337     ctl_t * ctl,
02338     atm_t * atm,
02339     double t);
02340
02341 #endif /* LIBTRAC_H */

```

5.25 met_conv.c File Reference

Convert file format of meteo data files.

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


Functions

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

5.25.1 Detailed Description

Convert file format of meteo data files.

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

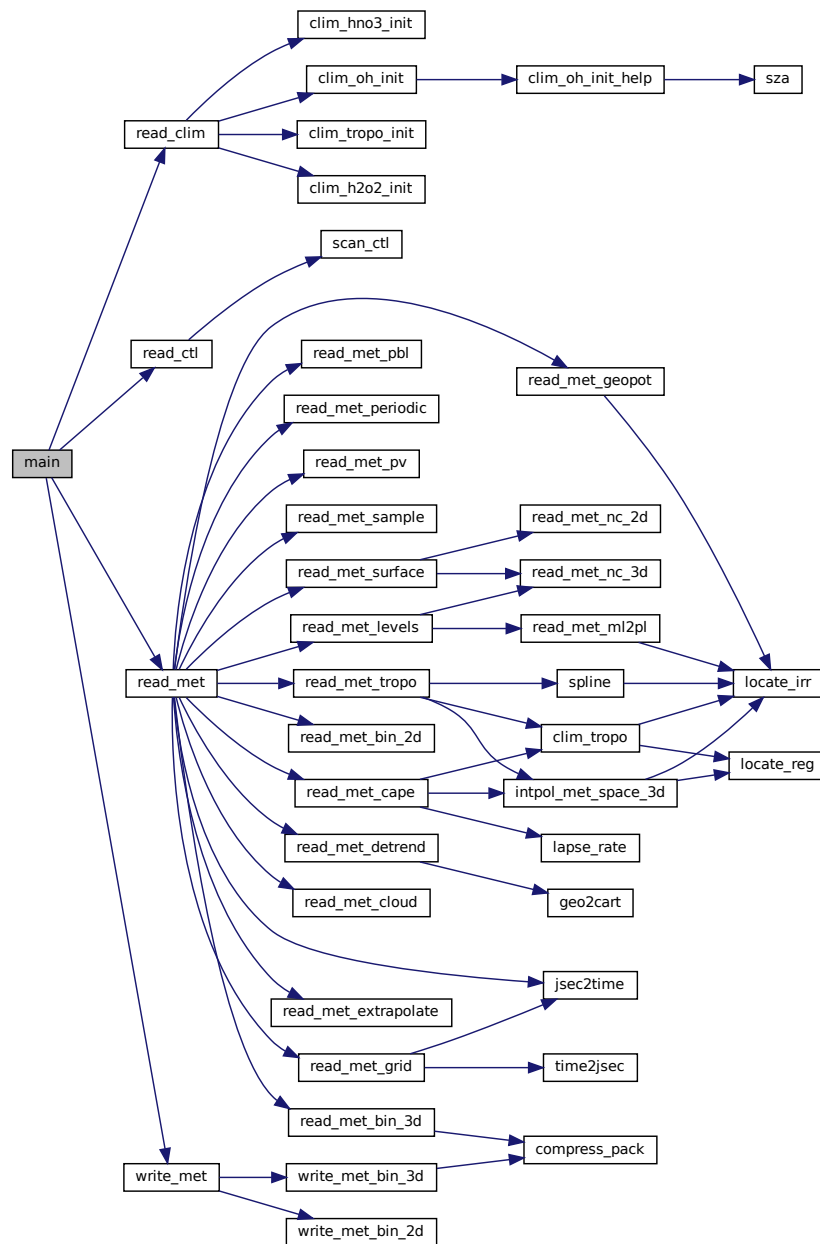
5.25.2 Function Documentation

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

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

```
00029     {
00030
00031     ctl_t ctl;
00032
00033     clim_t *clim;
00034
00035     met_t *met;
00036
00037     /* Check arguments... */
00038     if (argc < 6)
00039         ERRMSG("Give parameters: <ctl> <met_in> <met_in_type>"
00040              " <met_out> <met_out_type>");
00041
00042     /* Allocate... */
00043     ALLOC(clim, clim_t, 1);
00044     ALLOC(met, met_t, 1);
00045
00046     /* Read control parameters... */
00047     read_ctl(argv[1], argc, argv, &ctl);
00048
00049     /* Read climatological data... */
00050     read_clim(&ctl, clim);
00051
00052     /* Read meteo data... */
00053     ctl.met_type = atoi(argv[3]);
00054     if (!read_met(argv[2], &ctl, clim, met))
00055         ERRMSG("Cannot open file!");
00056
00057     /* Write meteo data... */
00058     ctl.met_type = atoi(argv[5]);
00059     write_met(argv[4], &ctl, met);
00060
00061     /* Free... */
00062     free(clim);
00063     free(met);
00064
00065     return EXIT_SUCCESS;
00066 }
```

Here is the call graph for this function:



5.26 met_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) 2013-2022 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      clim_t *clim;
00034
00035      met_t *met;
00036
00037      /* Check arguments... */
00038      if (argc < 6)
00039          ERRMSG("Give parameters: <ctl> <met_in> <met_in_type>"
00040               " <met_out> <met_out_type>");
00041
00042      /* Allocate... */
00043      ALLOC(clim, clim_t, 1);
00044      ALLOC(met, met_t, 1);
00045
00046      /* Read control parameters... */
00047      read_ctl(argv[1], argc, argv, &ctl);
00048
00049      /* Read climatological data... */
00050      read_clim(&ctl, clim);
00051
00052      /* Read meteo data... */
00053      ctl.met_type = atoi(argv[3]);
00054      if (!read_met(argv[2], &ctl, clim, met))
00055          ERRMSG("Cannot open file!");
00056
00057      /* Write meteo data... */
00058      ctl.met_type = atoi(argv[5]);
00059      write_met(argv[4], &ctl, met);
00060
00061      /* Free... */
00062      free(clim);
00063      free(met);
00064
00065      return EXIT_SUCCESS;
00066 }

```

5.27 met_lapse.c File Reference

Calculate lapse rate statistics.

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

Macros

- `#define LAPSEMIN -20.0`
Lapse rate minimum [K/km].
- `#define DLAPSE 0.1`
Lapse rate bin size [K/km].
- `#define IDXMAX 400`
Maximum number of histogram bins.

Functions

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

5.27.1 Detailed Description

Calculate lapse rate statistics.

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

5.27.2 Macro Definition Documentation

5.27.2.1 LAPSEMIN `#define LAPSEMIN -20.0`

Lapse rate minimum [K/km].

Definition at line 32 of file [met_lapse.c](#).

5.27.2.2 DLAPSE `#define DLAPSE 0.1`

Lapse rate bin size [K/km].

Definition at line 35 of file [met_lapse.c](#).

5.27.2.3 IDXMAX `#define IDXMAX 400`

Maximum number of histogram bins.

Definition at line 38 of file [met_lapse.c](#).

5.27.3 Function Documentation

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

Definition at line 44 of file [met_lapse.c](#).

```

00046     {
00047
00048     ctl_t ctl;
00049
00050     clim_t *clim;
00051
00052     met_t *met;
00053
00054     FILE *out;
00055
00056     static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00057         z_mean;
00058
00059     static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00060         nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <lapse.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00073     double lat0 =
00074         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00075     double lat1 =
00076         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00077     double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00078     double z1 =
00079         (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00080     int intpol =
00081         (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);
00082
00083     /* Read climatological data... */
00084     read_clim(&ctl, clim);
00085
00086     /* Loop over files... */
00087     for (int i = 3; i < argc; i++) {
00088
00089         /* Read meteorological data... */
00090         if (!read_met(argv[i], &ctl, clim, met))
00091             continue;
00092
00093         /* Get altitude and pressure profiles... */
00094         for (int iz = 0; iz < met->np; iz++)
00095             z[iz] = Z(met->p[iz]);
00096         for (int iz = 0; iz <= 250; iz++) {
00097             z2[iz] = 0.0 + 0.1 * iz;
00098             p2[iz] = P(z2[iz]);
00099         }
00100
00101         /* Loop over grid points... */
00102         for (int ix = 0; ix < met->nx; ix++)
00103             for (int iy = 0; iy < met->ny; iy++) {
00104
00105                 /* Check latitude range... */
00106                 if (met->lat[iy] < lat0 || met->lat[iy] > lat1)
00107                     continue;
00108
00109                 /* Interpolate temperature profile... */
00110                 for (int iz = 0; iz < met->np; iz++)
00111                     t[iz] = met->t[ix][iy][iz];
00112                 if (intpol == 1)
00113                     spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
00114                 else
00115                     for (int iz = 0; iz <= 250; iz++) {
00116                         int idx = locate_irr(z, met->np, z2[iz]);
00117                         t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
00118                     }
00119
00120                 /* Loop over vertical levels... */
00121                 for (int iz = 0; iz <= 250; iz++) {
00122
00123                     /* Check height range... */
00124                     if (z2[iz] < z0 || z2[iz] > z1)
00125                         continue;

```

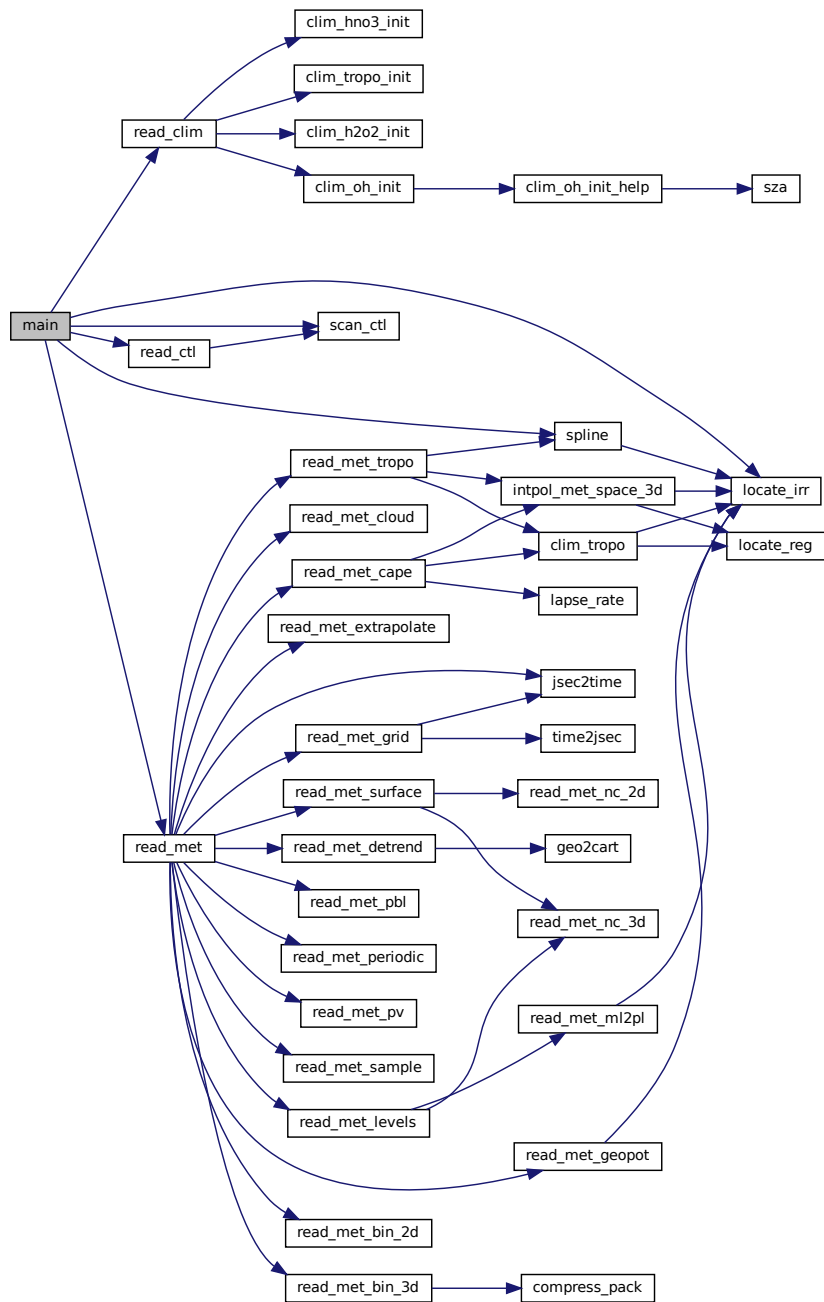
```

00126
00127     /* Check surface pressure... */
00128     if (p2[iz] > met->ps[ix][iy])
00129         continue;
00130
00131     /* Get mean latitude and height... */
00132     lat_mean += met->lat[iy];
00133     z_mean += z2[iz];
00134     np++;
00135
00136     /* Get lapse rates within a vertical layer... */
00137     int nlapse = 0;
00138     double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00139         0, lapse_sig = 0;
00140     for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00141         lapse_max =
00142             GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00143         lapse_min =
00144             GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00145         lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00146         lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00147         nlapse++;
00148     }
00149     lapse_mean /= nlapse;
00150     lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00151
00152     /* Get histograms... */
00153     int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00154     if (idx >= 0 && idx < IDXMAX) {
00155         hist_max[idx]++;
00156         nhist_max++;
00157     }
00158
00159     idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00160     if (idx >= 0 && idx < IDXMAX) {
00161         hist_min[idx]++;
00162         nhist_min++;
00163     }
00164
00165     idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00166     if (idx >= 0 && idx < IDXMAX) {
00167         hist_mean[idx]++;
00168         nhist_mean++;
00169     }
00170
00171     idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00172     if (idx >= 0 && idx < IDXMAX) {
00173         hist_sig[idx]++;
00174         nhist_sig++;
00175     }
00176 }
00177 }
00178 }
00179
00180 /* Create output file... */
00181 LOG(1, "Write lapse rate data: %s", argv[2]);
00182 if (!(out = fopen(argv[2], "w")))
00183     ERRMSG("Cannot create file!");
00184
00185 /* Write header... */
00186 fprintf(out,
00187     "# $1 = mean altitude [km]\n"
00188     "# $2 = mean latitude [deg]\n"
00189     "# $3 = lapse rate [K/km]\n"
00190     "# $4 = counts of maxima per bin\n"
00191     "# $5 = total number of maxima\n"
00192     "# $6 = normalized frequency of maxima\n"
00193     "# $7 = counts of minima per bin\n"
00194     "# $8 = total number of minima\n"
00195     "# $9 = normalized frequency of minima\n"
00196     "# $10 = counts of means per bin\n"
00197     "# $11 = total number of means\n"
00198     "# $12 = normalized frequency of means\n"
00199     "# $13 = counts of sigmas per bin\n"
00200     "# $14 = total number of sigmas\n"
00201     "# $15 = normalized frequency of sigmas\n\n");
00202
00203 /* Write data... */
00204 double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00205 for (int idx = 0; idx < IDXMAX; idx++) {
00206     nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00207     nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00208     nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00209     nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00210 }
00211 for (int idx = 0; idx < IDXMAX; idx++)
00212     fprintf(out,

```

```
00213         "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
00214         z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00215         hist_max[idx], nhist_max,
00216         (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00217         nhist_min, (double) hist_min[idx] / (double) nmax_min,
00218         hist_mean[idx], nhist_mean,
00219         (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00220         nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00221
00222     /* Close file... */
00223     fclose(out);
00224
00225     /* Free... */
00226     free(clim);
00227     free(met);
00228
00229     return EXIT_SUCCESS;
00230 }
```

Here is the call graph for this function:



5.28 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     clim_t *clim;
00051
00052     met_t *met;
00053
00054     FILE *out;
00055
00056     static double p2[1000], t[1000], t2[1000], z[1000], z2[1000], lat_mean,
00057         z_mean;
00058
00059     static int hist_max[1000], hist_min[1000], hist_mean[1000], hist_sig[1000],
00060         nhist_max, nhist_min, nhist_mean, nhist_sig, np;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <lapse.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     int dz = (int) scan_ctl(argv[1], argc, argv, "LAPSE_DZ", -1, "20", NULL);
00073     double lat0 =
00074         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT0", -1, "-90", NULL);
00075     double lat1 =
00076         (int) scan_ctl(argv[1], argc, argv, "LAPSE_LAT1", -1, "90", NULL);
00077     double z0 = (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z0", -1, "0", NULL);
00078     double z1 =
00079         (int) scan_ctl(argv[1], argc, argv, "LAPSE_Z1", -1, "100", NULL);
00080     int intpol =
00081         (int) scan_ctl(argv[1], argc, argv, "LAPSE_INTPOL", -1, "1", NULL);
00082
00083     /* Read climatological data... */
00084     read_clim(&ctl, clim);
00085
00086     /* Loop over files... */
00087     for (int i = 3; i < argc; i++) {
00088
00089         /* Read meteorological data... */
00090         if (!read_met(argv[i], &ctl, clim, met))
00091             continue;
00092
00093         /* Get altitude and pressure profiles... */
00094         for (int iz = 0; iz < met->np; iz++)
00095             z[iz] = Z(met->p[iz]);
00096         for (int iz = 0; iz <= 250; iz++) {
00097             z2[iz] = 0.0 + 0.1 * iz;
00098             p2[iz] = P(z2[iz]);
00099         }
00100
00101         /* Loop over grid points... */
00102         for (int ix = 0; ix < met->nx; ix++)
00103             for (int iy = 0; iy < met->ny; iy++) {
00104
00105                 /* Check latitude range... */
00106                 if (met->lat[iy] < lat0 || met->lat[iy] > lat1)

```

```

00107         continue;
00108
00109     /* Interpolate temperature profile... */
00110     for (int iz = 0; iz < met->np; iz++)
00111         t[iz] = met->t[ix][iy][iz];
00112     if (intpol == 1)
00113         spline(z, t, met->np, z2, t2, 251, ctl.met_tropo_spline);
00114     else
00115         for (int iz = 0; iz <= 250; iz++) {
00116             int idx = locate_irr(z, met->np, z2[iz]);
00117             t2[iz] = LIN(z[idx], t[idx], z[idx + 1], t[idx + 1], z2[iz]);
00118         }
00119
00120     /* Loop over vertical levels... */
00121     for (int iz = 0; iz <= 250; iz++) {
00122
00123         /* Check height range... */
00124         if (z2[iz] < z0 || z2[iz] > z1)
00125             continue;
00126
00127         /* Check surface pressure... */
00128         if (p2[iz] > met->ps[ix][iy])
00129             continue;
00130
00131         /* Get mean latitude and height... */
00132         lat_mean += met->lat[iy];
00133         z_mean += z2[iz];
00134         np++;
00135
00136         /* Get lapse rates within a vertical layer... */
00137         int nlapse = 0;
00138         double lapse_max = -1e99, lapse_min = 1e99, lapse_mean =
00139             0, lapse_sig = 0;
00140         for (int iz2 = iz + 1; iz2 <= iz + dz; iz2++) {
00141             lapse_max =
00142                 GSL_MAX(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_max);
00143             lapse_min =
00144                 GSL_MIN(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]), lapse_min);
00145             lapse_mean += LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]);
00146             lapse_sig += SQR(LAPSE(p2[iz], t2[iz], p2[iz2], t2[iz2]));
00147             nlapse++;
00148         }
00149         lapse_mean /= nlapse;
00150         lapse_sig = sqrt(GSL_MAX(lapse_sig / nlapse - SQR(lapse_mean), 0));
00151
00152         /* Get histograms... */
00153         int idx = (int) ((lapse_max - LAPSEMIN) / DLAPSE);
00154         if (idx >= 0 && idx < IDXMAX) {
00155             hist_max[idx]++;
00156             nhist_max++;
00157         }
00158
00159         idx = (int) ((lapse_min - LAPSEMIN) / DLAPSE);
00160         if (idx >= 0 && idx < IDXMAX) {
00161             hist_min[idx]++;
00162             nhist_min++;
00163         }
00164
00165         idx = (int) ((lapse_mean - LAPSEMIN) / DLAPSE);
00166         if (idx >= 0 && idx < IDXMAX) {
00167             hist_mean[idx]++;
00168             nhist_mean++;
00169         }
00170
00171         idx = (int) ((lapse_sig - LAPSEMIN) / DLAPSE);
00172         if (idx >= 0 && idx < IDXMAX) {
00173             hist_sig[idx]++;
00174             nhist_sig++;
00175         }
00176     }
00177 }
00178 }
00179
00180 /* Create output file... */
00181 LOG(1, "Write lapse rate data: %s", argv[2]);
00182 if (!(out = fopen(argv[2], "w")))
00183     ERRMSG("Cannot create file!");
00184
00185 /* Write header... */
00186 fprintf(out,
00187     "# $1 = mean altitude [km]\n"
00188     "# $2 = mean latitude [deg]\n"
00189     "# $3 = lapse rate [K/km]\n"
00190     "# $4 = counts of maxima per bin\n"
00191     "# $5 = total number of maxima\n"
00192     "# $6 = normalized frequency of maxima\n"
00193     "# $7 = counts of minima per bin\n"

```

```

00194         "# $8 = total number of minima\n"
00195         "# $9 = normalized frequency of minima\n"
00196         "# $10 = counts of means per bin\n"
00197         "# $11 = total number of means\n"
00198         "# $12 = normalized frequency of means\n"
00199         "# $13 = counts of sigmas per bin\n"
00200         "# $14 = total number of sigmas\n"
00201         "# $15 = normalized frequency of sigmas\n\n");
00202
00203     /* Write data... */
00204     double nmax_max = 0, nmax_min = 0, nmax_mean = 0, nmax_sig = 0;
00205     for (int idx = 0; idx < IDXMAX; idx++) {
00206         nmax_max = GSL_MAX(hist_max[idx], nmax_max);
00207         nmax_min = GSL_MAX(hist_min[idx], nmax_min);
00208         nmax_mean = GSL_MAX(hist_mean[idx], nmax_mean);
00209         nmax_sig = GSL_MAX(hist_sig[idx], nmax_sig);
00210     }
00211     for (int idx = 0; idx < IDXMAX; idx++)
00212         fprintf(out,
00213             "%g %g %g %d %d %g %d %d %g %d %d %g %d %d %g\n",
00214             z_mean / np, lat_mean / np, (idx + .5) * DLAPSE + LAPSEMIN,
00215             hist_max[idx], nhist_max,
00216             (double) hist_max[idx] / (double) nmax_max, hist_min[idx],
00217             nhist_min, (double) hist_min[idx] / (double) nmax_min,
00218             hist_mean[idx], nhist_mean,
00219             (double) hist_mean[idx] / (double) nmax_mean, hist_sig[idx],
00220             nhist_sig, (double) hist_sig[idx] / (double) nmax_sig);
00221
00222     /* Close file... */
00223     fclose(out);
00224
00225     /* Free... */
00226     free(clim);
00227     free(met);
00228
00229     return EXIT_SUCCESS;
00230 }

```

5.29 met_map.c File Reference

Extract map from meteorological data.

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

Macros

- `#define NX 1441`
Maximum number of longitudes.
- `#define NY 721`
Maximum number of latitudes.

Functions

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

5.29.1 Detailed Description

Extract map from meteorological data.

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

5.29.2 Macro Definition Documentation

5.29.2.1 NX #define NX 1441

Maximum number of longitudes.

Definition at line 32 of file [met_map.c](#).

5.29.2.2 NY #define NY 721

Maximum number of latitudes.

Definition at line 35 of file [met_map.c](#).

5.29.3 Function Documentation

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

Definition at line 41 of file [met_map.c](#).

```

00043     {
00044
00045     ctl_t ctl;
00046
00047     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
00054     zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
00055     ptm[NX][NY], t, pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY],
00056     w, wm[NX][NY], h2o, h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00057     hno3m[NX][NY], ohm[NX][NY], h2o2m[NX][NY], tdewm[NX][NY], ticem[NX][NY],
00058     tnatm[NX][NY], lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
00059     pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY], pct, pctm[NX][NY], pcb,
00060     pcbm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
00061     pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
00062     rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
00063     lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00064
00065     static int i, ix, iy, np[NX][NY], npc[NX][NY], npt[NX][NY], nx, ny, ci[3];
00066
00067     /* Allocate... */
00068     ALLOC(clim, clim_t, 1);
00069     ALLOC(met, met_t, 1);
00070
00071     /* Check arguments... */
00072     if (argc < 4)
00073         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00074
00075     /* Read control parameters... */
00076     read_ctl(argv[1], argc, argv, &ctl);
00077     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00078     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00079     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00080     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00081     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);

```

```

00082 lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00083 dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00084 theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00085
00086 /* Read climatological data... */
00087 read_clim(&ctl, clim);
00088
00089 /* Loop over files... */
00090 for (i = 3; i < argc; i++) {
00091
00092     /* Read meteorological data... */
00093     if (!read_met(argv[i], &ctl, clim, met))
00094         continue;
00095
00096     /* Set horizontal grid... */
00097     if (dlon <= 0)
00098         dlon = fabs(met->lon[1] - met->lon[0]);
00099     if (dlat <= 0)
00100         dlat = fabs(met->lat[1] - met->lat[0]);
00101     if (lon0 < -360 && lon1 > 360) {
00102         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00103         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00104     }
00105     nx = ny = 0;
00106     for (lon = lon0; lon <= lon1; lon += dlon) {
00107         lons[nx] = lon;
00108         if (++nx > NX)
00109             ERRMSG("Too many longitudes!");
00110     }
00111     if (lat0 < -90 && lat1 > 90) {
00112         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00113         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00114     }
00115     for (lat = lat0; lat <= lat1; lat += dlat) {
00116         lats[ny] = lat;
00117         if (++ny > NY)
00118             ERRMSG("Too many latitudes!");
00119     }
00120
00121     /* Average... */
00122     for (ix = 0; ix < nx; ix++)
00123         for (iy = 0; iy < ny; iy++) {
00124
00125             /* Find pressure level for given theta level... */
00126             if (theta > 0) {
00127                 ptop = met->p[met->np - 1];
00128                 pbot = met->p[0];
00129                 do {
00130                     p0 = 0.5 * (ptop + pbot);
00131                     intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00132                                         &t0, ci, cw, 1);
00133                     if (THETA(p0, t0) > theta)
00134                         ptop = p0;
00135                     else
00136                         pbot = p0;
00137                 } while (fabs(ptop - pbot) > 1e-5);
00138             }
00139
00140             /* Interpolate meteo data... */
00141             INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00142
00143             /* Averaging... */
00144             timem[ix][iy] += met->time;
00145             zm[ix][iy] += z;
00146             pm[ix][iy] += p0;
00147             tm[ix][iy] += t;
00148             um[ix][iy] += u;
00149             vm[ix][iy] += v;
00150             wm[ix][iy] += w;
00151             pvm[ix][iy] += pv;
00152             h2om[ix][iy] += h2o;
00153             o3m[ix][iy] += o3;
00154             lwcm[ix][iy] += lwc;
00155             iwcm[ix][iy] += iwc;
00156             psm[ix][iy] += ps;
00157             tsm[ix][iy] += ts;
00158             zsm[ix][iy] += zs;
00159             usm[ix][iy] += us;
00160             vsm[ix][iy] += vs;
00161             pblm[ix][iy] += pbl;
00162             pctm[ix][iy] += pct;
00163             pcbm[ix][iy] += pcb;
00164             clm[ix][iy] += cl;
00165             if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00166                 && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00167                 plclm[ix][iy] += plcl;
00168                 plfcm[ix][iy] += plfc;

```

```

00169         pelm[ix][iy] += pel;
00170         capem[ix][iy] += cape;
00171         cinm[ix][iy] += cin;
00172         npc[ix][iy]++;
00173     }
00174     if (gsl_finite(pt)) {
00175         ptm[ix][iy] += pt;
00176         ztm[ix][iy] += zt;
00177         ttm[ix][iy] += tt;
00178         h2otm[ix][iy] += h2ot;
00179         npt[ix][iy]++;
00180     }
00181     hno3m[ix][iy] += clim_hno3(clim, met->time, lats[iy], p0);
00182     tnatm[ix][iy] +=
00183         nat_temperature(p0, h2o, clim_hno3(clim, met->time, lats[iy], p0));
00184     ohm[ix][iy] +=
00185         clim_oh_diurnal(&ctl, clim, met->time, p0, lons[ix], lats[iy]);
00186     h2o2m[ix][iy] += clim_h2o2(clim, met->time, lats[iy], p0);
00187     rhm[ix][iy] += RH(p0, t, h2o);
00188     rhicem[ix][iy] += RHICE(p0, t, h2o);
00189     tdewm[ix][iy] += TDEW(p0, h2o);
00190     ticem[ix][iy] += TICE(p0, h2o);
00191     np[ix][iy]++;
00192 }
00193 }
00194
00195 /* Create output file... */
00196 LOG(1, "Write meteorological data file: %s", argv[2]);
00197 if (!out = fopen(argv[2], "w"))
00198     ERRMSG("Cannot create file!");
00199
00200 /* Write header... */
00201 fprintf(out,
00202     "# $1 = time [s]\n"
00203     "# $2 = altitude [km]\n"
00204     "# $3 = longitude [deg]\n"
00205     "# $4 = latitude [deg]\n"
00206     "# $5 = pressure [hPa]\n"
00207     "# $6 = temperature [K]\n"
00208     "# $7 = zonal wind [m/s]\n"
00209     "# $8 = meridional wind [m/s]\n"
00210     "# $9 = vertical velocity [hPa/s]\n"
00211     "# $10 = H2O volume mixing ratio [ppv]\n");
00212 fprintf(out,
00213     "# $11 = O3 volume mixing ratio [ppv]\n"
00214     "# $12 = geopotential height [km]\n"
00215     "# $13 = potential vorticity [PVU]\n"
00216     "# $14 = surface pressure [hPa]\n"
00217     "# $15 = surface temperature [K]\n"
00218     "# $16 = surface geopotential height [km]\n"
00219     "# $17 = surface zonal wind [m/s]\n"
00220     "# $18 = surface meridional wind [m/s]\n"
00221     "# $19 = tropopause pressure [hPa]\n"
00222     "# $20 = tropopause geopotential height [km]\n");
00223 fprintf(out,
00224     "# $21 = tropopause temperature [K]\n"
00225     "# $22 = tropopause water vapor [ppv]\n"
00226     "# $23 = cloud liquid water content [kg/kg]\n"
00227     "# $24 = cloud ice water content [kg/kg]\n"
00228     "# $25 = total column cloud water [kg/m^2]\n"
00229     "# $26 = cloud top pressure [hPa]\n"
00230     "# $27 = cloud bottom pressure [hPa]\n"
00231     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00232     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00233     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00234 fprintf(out,
00235     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00236     "# $32 = convective inhibition (CIN) [J/kg]\n"
00237     "# $33 = relative humidity over water [%]\n"
00238     "# $34 = relative humidity over ice [%]\n"
00239     "# $35 = dew point temperature [K]\n"
00240     "# $36 = frost point temperature [K]\n"
00241     "# $37 = NAT temperature [K]\n"
00242     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00243     "# $39 = OH concentration [molec/cm^3]\n"
00244     "# $40 = H2O2 concentration [molec/cm^3]\n");
00245 fprintf(out,
00246     "# $41 = boundary layer pressure [hPa]\n"
00247     "# $42 = number of data points\n"
00248     "# $43 = number of tropopause data points\n"
00249     "# $44 = number of CAPE data points\n");
00250
00251 /* Write data... */
00252 for (iy = 0; iy < ny; iy++) {
00253     fprintf(out, "\n");
00254     for (ix = 0; ix < nx; ix++)
00255         fprintf(out,

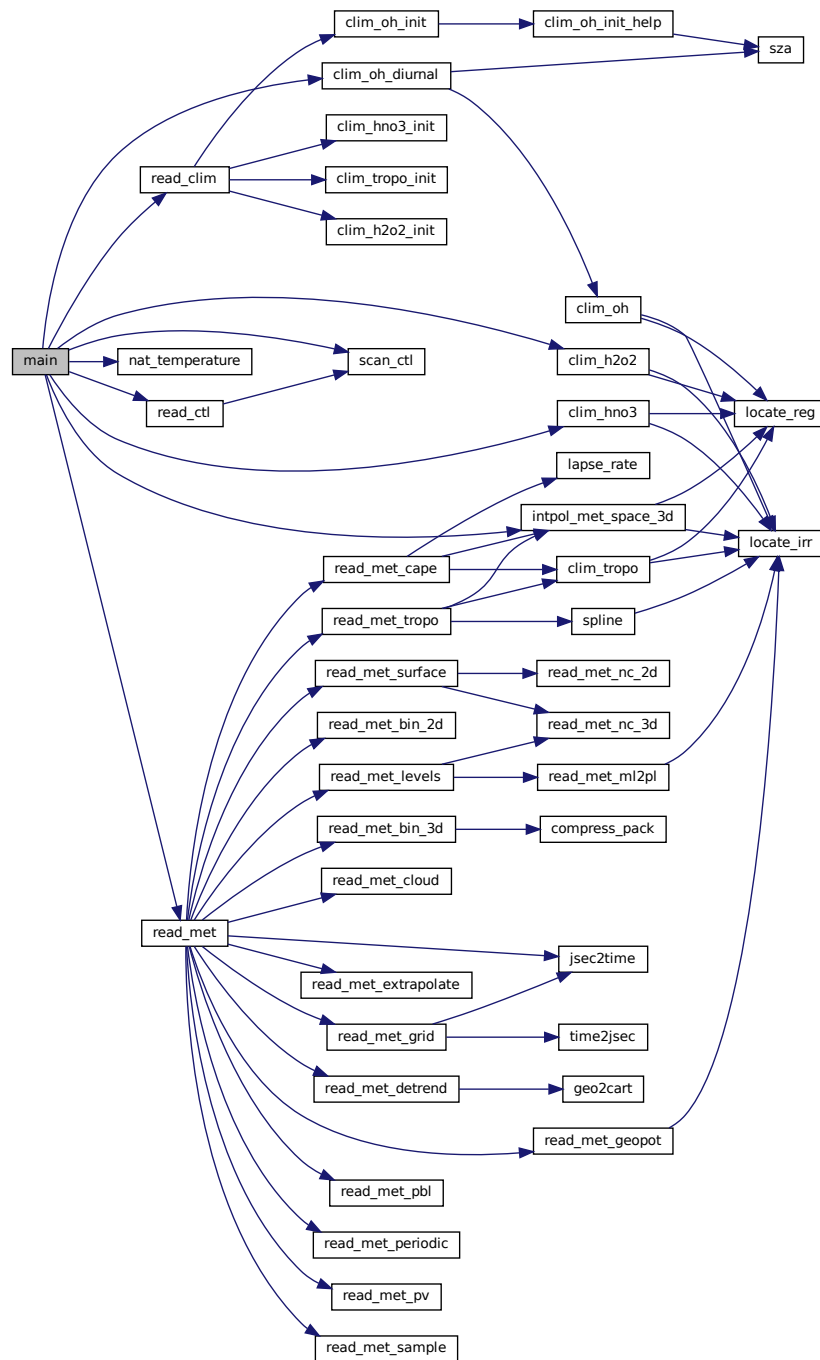
```

```

00256         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g"
00257         " %g %g %g %g %g %g %g %g %g %g %g %g %g %g"
00258         " %g %g %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
00259         timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00260         lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00261         tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00262         vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00263         h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00264         zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00265         psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00266         zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00267         vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
00268         ztm[ix][iy] / npt[ix][iy], ttm[ix][iy] / npt[ix][iy],
00269         h2otm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
00270         iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00271         pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
00272         plclm[ix][iy] / npc[ix][iy], plfcm[ix][iy] / npc[ix][iy],
00273         pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
00274         cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
00275         rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
00276         ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
00277         hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
00278         h2o2m[ix][iy] / np[ix][iy], pblm[ix][iy] / np[ix][iy],
00279         np[ix][iy], npt[ix][iy], npc[ix][iy]);
00280     }
00281
00282     /* Close file... */
00283     fclose(out);
00284
00285     /* Free... */
00286     free(clim);
00287     free(met);
00288
00289     return EXIT_SUCCESS;
00290 }

```

Here is the call graph for this function:



5.30 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) 2013-2022 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  clim_t *clim;
00048
00049  met_t *met;
00050
00051  FILE *out;
00052
00053  static double timem[NX][NY], p0, ps, psm[NX][NY], ts, tsm[NX][NY], zs,
00054  zsm[NX][NY], us, usm[NX][NY], vs, vsm[NX][NY], pbl, pblm[NX][NY], pt,
00055  ptm[NX][NY], t, pm[NX][NY], tm[NX][NY], u, um[NX][NY], v, vm[NX][NY],
00056  w, wm[NX][NY], h2o, h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00057  hno3m[NX][NY], ohm[NX][NY], h2o2m[NX][NY], tdewm[NX][NY], ticem[NX][NY],
00058  tnatm[NX][NY], lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
00059  pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY], pct, pctm[NX][NY], pcb,
00060  pcbm[NX][NY], cl, clm[NX][NY], plcl, plclm[NX][NY], plfc, plfcm[NX][NY],
00061  pel, pelm[NX][NY], cape, capem[NX][NY], cin, cinm[NX][NY],
00062  rhm[NX][NY], rhicem[NX][NY], theta, ptop, pbot, t0,
00063  lon, lon0, lon1, lons[NX], dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00064
00065  static int i, ix, iy, np[NX][NY], npc[NX][NY], npt[NX][NY], nx, ny, ci[3];
00066
00067  /* Allocate... */
00068  ALLOC(clim, clim_t, 1);
00069  ALLOC(met, met_t, 1);
00070
00071  /* Check arguments... */
00072  if (argc < 4)
00073  ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00074
00075  /* Read control parameters... */
00076  read_ctl(argv[1], argc, argv, &ctl);
00077  p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00078  lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00079  lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00080  dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00081  lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00082  lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00083  dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00084  theta = scan_ctl(argv[1], argc, argv, "MAP_THETA", -1, "-999", NULL);
00085
00086  /* Read climatological data... */
00087  read_clim(&ctl, clim);
00088
00089  /* Loop over files... */
00090  for (i = 3; i < argc; i++) {
00091
00092  /* Read meteorological data... */
00093  if (!read_met(argv[i], &ctl, clim, met))
00094  continue;
00095
00096  /* Set horizontal grid... */
00097  if (dlon <= 0)
00098  dlon = fabs(met->lon[1] - met->lon[0]);
00099  if (dlat <= 0)
00100  dlat = fabs(met->lat[1] - met->lat[0]);
00101  if (lon0 < -360 && lon1 > 360) {
00102  lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);

```

```

00103     lonl = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00104 }
00105 nx = ny = 0;
00106 for (lon = lon0; lon <= lonl; lon += dlon) {
00107     lons[nx] = lon;
00108     if (++nx > NX)
00109         ERRMSG("Too many longitudes!");
00110 }
00111 if (lat0 < -90 && latl > 90) {
00112     lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00113     latl = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00114 }
00115 for (lat = lat0; lat <= latl; lat += dlat) {
00116     lats[ny] = lat;
00117     if (++ny > NY)
00118         ERRMSG("Too many latitudes!");
00119 }
00120
00121 /* Average... */
00122 for (ix = 0; ix < nx; ix++)
00123     for (iy = 0; iy < ny; iy++) {
00124
00125         /* Find pressure level for given theta level... */
00126         if (theta > 0) {
00127             ptop = met->p[met->np - 1];
00128             pbot = met->p[0];
00129             do {
00130                 p0 = 0.5 * (ptop + pbot);
00131                 intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy],
00132                                     &t0, ci, cw, 1);
00133                 if (THETA(p0, t0) > theta)
00134                     ptop = p0;
00135                 else
00136                     pbot = p0;
00137             } while (fabs(ptop - pbot) > 1e-5);
00138         }
00139
00140         /* Interpolate meteo data... */
00141         INTPOL_SPACE_ALL(p0, lons[ix], lats[iy]);
00142
00143         /* Averaging... */
00144         timem[ix][iy] += met->time;
00145         zm[ix][iy] += z;
00146         pm[ix][iy] += p0;
00147         tm[ix][iy] += t;
00148         um[ix][iy] += u;
00149         vm[ix][iy] += v;
00150         wm[ix][iy] += w;
00151         pvm[ix][iy] += pv;
00152         h2om[ix][iy] += h2o;
00153         o3m[ix][iy] += o3;
00154         lwcm[ix][iy] += lwc;
00155         iwcm[ix][iy] += iwc;
00156         psm[ix][iy] += ps;
00157         tsm[ix][iy] += ts;
00158         zsm[ix][iy] += zs;
00159         usm[ix][iy] += us;
00160         vsm[ix][iy] += vs;
00161         pblm[ix][iy] += pbl;
00162         pctm[ix][iy] += pct;
00163         pcbm[ix][iy] += pcb;
00164         clm[ix][iy] += cl;
00165         if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00166             && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00167             plclm[ix][iy] += plcl;
00168             plfcm[ix][iy] += plfc;
00169             pelm[ix][iy] += pel;
00170             capem[ix][iy] += cape;
00171             cinm[ix][iy] += cin;
00172             npc[ix][iy]++;
00173         }
00174         if (gsl_finite(pt)) {
00175             ptm[ix][iy] += pt;
00176             ztm[ix][iy] += zt;
00177             ttm[ix][iy] += tt;
00178             h2otm[ix][iy] += h2ot;
00179             npt[ix][iy]++;
00180         }
00181         hno3m[ix][iy] += clim_hno3(clim, met->time, lats[iy], p0);
00182         tnatm[ix][iy] +=
00183             nat_temperature(p0, h2o, clim_hno3(clim, met->time, lats[iy], p0));
00184         ohm[ix][iy] +=
00185             clim_oh_diurnal(&ctl, clim, met->time, p0, lons[ix], lats[iy]);
00186         h2o2m[ix][iy] += clim_h2o2(clim, met->time, lats[iy], p0);
00187         rhm[ix][iy] += RH(p0, t, h2o);
00188         rhicem[ix][iy] += RHICE(p0, t, h2o);
00189         tdewm[ix][iy] += TDEW(p0, h2o);

```

```
00190 ticem[ix][iy] += TICE(p0, h2o);
00191 np[ix][iy]++;
00192 }
00193 }
00194
00195 /* Create output file... */
00196 LOG(1, "Write meteorological data file: %s", argv[2]);
00197 if (!out = fopen(argv[2], "w"))
00198     ERRMSG("Cannot create file!");
00199
00200 /* Write header... */
00201 fprintf(out,
00202         "# $1 = time [s]\n"
00203         "# $2 = altitude [km]\n"
00204         "# $3 = longitude [deg]\n"
00205         "# $4 = latitude [deg]\n"
00206         "# $5 = pressure [hPa]\n"
00207         "# $6 = temperature [K]\n"
00208         "# $7 = zonal wind [m/s]\n"
00209         "# $8 = meridional wind [m/s]\n"
00210         "# $9 = vertical velocity [hPa/s]\n"
00211         "# $10 = H2O volume mixing ratio [ppv]\n");
00212 fprintf(out,
00213         "# $11 = O3 volume mixing ratio [ppv]\n"
00214         "# $12 = geopotential height [km]\n"
00215         "# $13 = potential vorticity [PVU]\n"
00216         "# $14 = surface pressure [hPa]\n"
00217         "# $15 = surface temperature [K]\n"
00218         "# $16 = surface geopotential height [km]\n"
00219         "# $17 = surface zonal wind [m/s]\n"
00220         "# $18 = surface meridional wind [m/s]\n"
00221         "# $19 = tropopause pressure [hPa]\n"
00222         "# $20 = tropopause geopotential height [km]\n");
00223 fprintf(out,
00224         "# $21 = tropopause temperature [K]\n"
00225         "# $22 = tropopause water vapor [ppv]\n"
00226         "# $23 = cloud liquid water content [kg/kg]\n"
00227         "# $24 = cloud ice water content [kg/kg]\n"
00228         "# $25 = total column cloud water [kg/m^2]\n"
00229         "# $26 = cloud top pressure [hPa]\n"
00230         "# $27 = cloud bottom pressure [hPa]\n"
00231         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00232         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00233         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00234 fprintf(out,
00235         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00236         "# $32 = convective inhibition (CIN) [J/kg]\n"
00237         "# $33 = relative humidity over water [%]\n"
00238         "# $34 = relative humidity over ice [%]\n"
00239         "# $35 = dew point temperature [K]\n"
00240         "# $36 = frost point temperature [K]\n"
00241         "# $37 = NAT temperature [K]\n"
00242         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00243         "# $39 = OH concentration [molec/cm^3]\n"
00244         "# $40 = H2O2 concentration [molec/cm^3]\n");
00245 fprintf(out,
00246         "# $41 = boundary layer pressure [hPa]\n"
00247         "# $42 = number of data points\n"
00248         "# $43 = number of tropopause data points\n"
00249         "# $44 = number of CAPE data points\n");
00250
00251 /* Write data... */
00252 for (iy = 0; iy < ny; iy++) {
00253     fprintf(out, "\n");
00254     for (ix = 0; ix < nx; ix++)
00255         fprintf(out,
00256                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n"
00257                 " %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00258                 " %g %g %g %g %g %g %g %g %g %g %g %g %d %d %d\n",
00259                 timem[ix][iy] / np[ix][iy], Z(pm[ix][iy] / np[ix][iy]),
00260                 lons[ix], lats[iy], pm[ix][iy] / np[ix][iy],
00261                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00262                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00263                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00264                 zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00265                 psm[ix][iy] / np[ix][iy], tsm[ix][iy] / np[ix][iy],
00266                 zsm[ix][iy] / np[ix][iy], usm[ix][iy] / np[ix][iy],
00267                 vsm[ix][iy] / np[ix][iy], ptm[ix][iy] / npt[ix][iy],
00268                 ztm[ix][iy] / npt[ix][iy], ttcm[ix][iy] / npt[ix][iy],
00269                 h2otcm[ix][iy] / npt[ix][iy], lwcm[ix][iy] / np[ix][iy],
00270                 iwcm[ix][iy] / np[ix][iy], clcm[ix][iy] / np[ix][iy],
00271                 pctm[ix][iy] / np[ix][iy], pcbm[ix][iy] / np[ix][iy],
00272                 plclm[ix][iy] / npc[ix][iy], plfc[m[ix][iy]] / npc[ix][iy],
00273                 pelm[ix][iy] / npc[ix][iy], capem[ix][iy] / npc[ix][iy],
00274                 cinm[ix][iy] / npc[ix][iy], rhm[ix][iy] / np[ix][iy],
00275                 rhicem[ix][iy] / np[ix][iy], tdewm[ix][iy] / np[ix][iy],
00276                 ticem[ix][iy] / np[ix][iy], tnatm[ix][iy] / np[ix][iy],
```

```
00277             hno3m[ix][iy] / np[ix][iy], ohm[ix][iy] / np[ix][iy],
00278             h2o2m[ix][iy] / np[ix][iy], pblm[ix][iy] / np[ix][iy],
00279             np[ix][iy], npt[ix][iy], npc[ix][iy]);
00280     }
00281
00282     /* Close file... */
00283     fclose(out);
00284
00285     /* Free... */
00286     free(clim);
00287     free(met);
00288
00289     return EXIT_SUCCESS;
00290 }
```

5.31 met_prof.c File Reference

Extract vertical profile from meteorological data.

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

Macros

- `#define NZ 1000`
Maximum number of altitudes.

Functions

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

5.31.1 Detailed Description

Extract vertical profile from meteorological data.

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

5.31.2 Macro Definition Documentation

5.31.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line [32](#) of file [met_prof.c](#).

5.31.3 Function Documentation

```

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

```

Definition at line 38 of file [met_prof.c](#).

```

00040     {
00041
00042     ctl_t ctl;
00043
00044     clim_t *clim;
00045
00046     met_t *met;
00047
00048     FILE *out;
00049
00050     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00051     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00052     wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00053     iwc, iwcm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00054     vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
00055     cl, clim[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
00056     cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
00057     pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
00058     tnatm[NZ], hno3m[NZ], ohm[NZ], h2o2m[NZ], cw[3];
00059
00060     static int i, iz, np[NZ], npc[NZ], npt[NZ], nz, ci[3];
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00073     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00074     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00075     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00076     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00077     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00078     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00079     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00080     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00081
00082     /* Read climatological data... */
00083     read_clim(&ctl, clim);
00084
00085     /* Loop over input files... */
00086     for (i = 3; i < argc; i++) {
00087
00088         /* Read meteorological data... */
00089         if (!read_met(argv[i], &ctl, clim, met))
00090             continue;
00091
00092         /* Set vertical grid... */
00093         if (z0 < 0)
00094             z0 = Z(met->p[0]);
00095         if (z1 < 0)
00096             z1 = Z(met->p[met->np - 1]);
00097         nz = 0;
00098         if (dz < 0) {
00099             for (iz = 0; iz < met->np; iz++)
00100                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00101                     plev[nz] = met->p[iz];
00102                     if (++nz > NZ)
00103                         ERRMSG("Too many pressure levels!");
00104                 }
00105             } else
00106                 for (z = z0; z <= z1; z += dz) {
00107                     plev[nz] = P(z);
00108                     if (++nz > NZ)
00109                         ERRMSG("Too many pressure levels!");
00110                 }
00111
00112         /* Set horizontal grid... */
00113         if (dlon <= 0)
00114             dlon = fabs(met->lon[1] - met->lon[0]);
00115         if (dlat <= 0)
00116             dlat = fabs(met->lat[1] - met->lat[0]);
00117
00118         /* Average... */
00119         for (iz = 0; iz < nz; iz++)

```

```

00120     for (lon = lon0; lon <= lon1; lon += dlon)
00121         for (lat = lat0; lat <= lat1; lat += dlat) {
00122
00123             /* Interpolate meteo data... */
00124             INTPOL_SPACE_ALL(plev[iz], lon, lat);
00125
00126             /* Averaging... */
00127             if (gsl_finite(t) && gsl_finite(u)
00128                 && gsl_finite(v) && gsl_finite(w)) {
00129                 timem[iz] += met->time;
00130                 lonm[iz] += lon;
00131                 latm[iz] += lat;
00132                 zm[iz] += z;
00133                 tm[iz] += t;
00134                 um[iz] += u;
00135                 vm[iz] += v;
00136                 wm[iz] += w;
00137                 pvm[iz] += pv;
00138                 h2om[iz] += h2o;
00139                 o3m[iz] += o3;
00140                 lwcm[iz] += lwc;
00141                 iwcm[iz] += iwc;
00142                 psm[iz] += ps;
00143                 tsm[iz] += ts;
00144                 zsm[iz] += zs;
00145                 usm[iz] += us;
00146                 vsm[iz] += vs;
00147                 pblm[iz] += pbl;
00148                 pctm[iz] += pct;
00149                 pcbm[iz] += pcb;
00150                 clm[iz] += cl;
00151                 if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00152                     && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00153                     plclm[iz] += plcl;
00154                     plfcm[iz] += plfc;
00155                     pelm[iz] += pel;
00156                     capem[iz] += cape;
00157                     cinm[iz] += cin;
00158                     npc[iz]++;
00159                 }
00160                 if (gsl_finite(pt)) {
00161                     ptm[iz] += pt;
00162                     ztm[iz] += zt;
00163                     ttm[iz] += tt;
00164                     h2otm[iz] += h2ot;
00165                     npt[iz]++;
00166                 }
00167                 rhm[iz] += RH(plev[iz], t, h2o);
00168                 rhicem[iz] += RHICE(plev[iz], t, h2o);
00169                 tdewm[iz] += TDEW(plev[iz], h2o);
00170                 ticem[iz] += TICE(plev[iz], h2o);
00171                 hno3m[iz] += clim_hno3(clim, met->time, lat, plev[iz]);
00172                 tnatm[iz] +=
00173                     nat_temperature(plev[iz], h2o,
00174                                     clim_hno3(clim, met->time, lat, plev[iz]));
00175                 ohm[iz] +=
00176                     clim_oh_diurnal(&ctl, clim, met->time, plev[iz], lon, lat);
00177                 h2o2m[iz] += clim_h2o2(clim, met->time, lat, plev[iz]);
00178                 np[iz]++;
00179             }
00180         }
00181     }
00182
00183     /* Create output file... */
00184     LOG(1, "Write meteorological data file: %s", argv[2]);
00185     if (!(out = fopen(argv[2], "w")))
00186         ERRMSG("Cannot create file!");
00187
00188     /* Write header... */
00189     fprintf(out,
00190         "# $1 = time [s]\n"
00191         "# $2 = altitude [km]\n"
00192         "# $3 = longitude [deg]\n"
00193         "# $4 = latitude [deg]\n"
00194         "# $5 = pressure [hPa]\n"
00195         "# $6 = temperature [K]\n"
00196         "# $7 = zonal wind [m/s]\n"
00197         "# $8 = meridional wind [m/s]\n"
00198         "# $9 = vertical velocity [hPa/s]\n"
00199         "# $10 = H2O volume mixing ratio [ppv]\n");
00200     fprintf(out,
00201         "# $11 = O3 volume mixing ratio [ppv]\n"
00202         "# $12 = geopotential height [km]\n"
00203         "# $13 = potential vorticity [PVU]\n"
00204         "# $14 = surface pressure [hPa]\n"
00205         "# $15 = surface temperature [K]\n"
00206         "# $16 = surface geopotential height [km]\n");

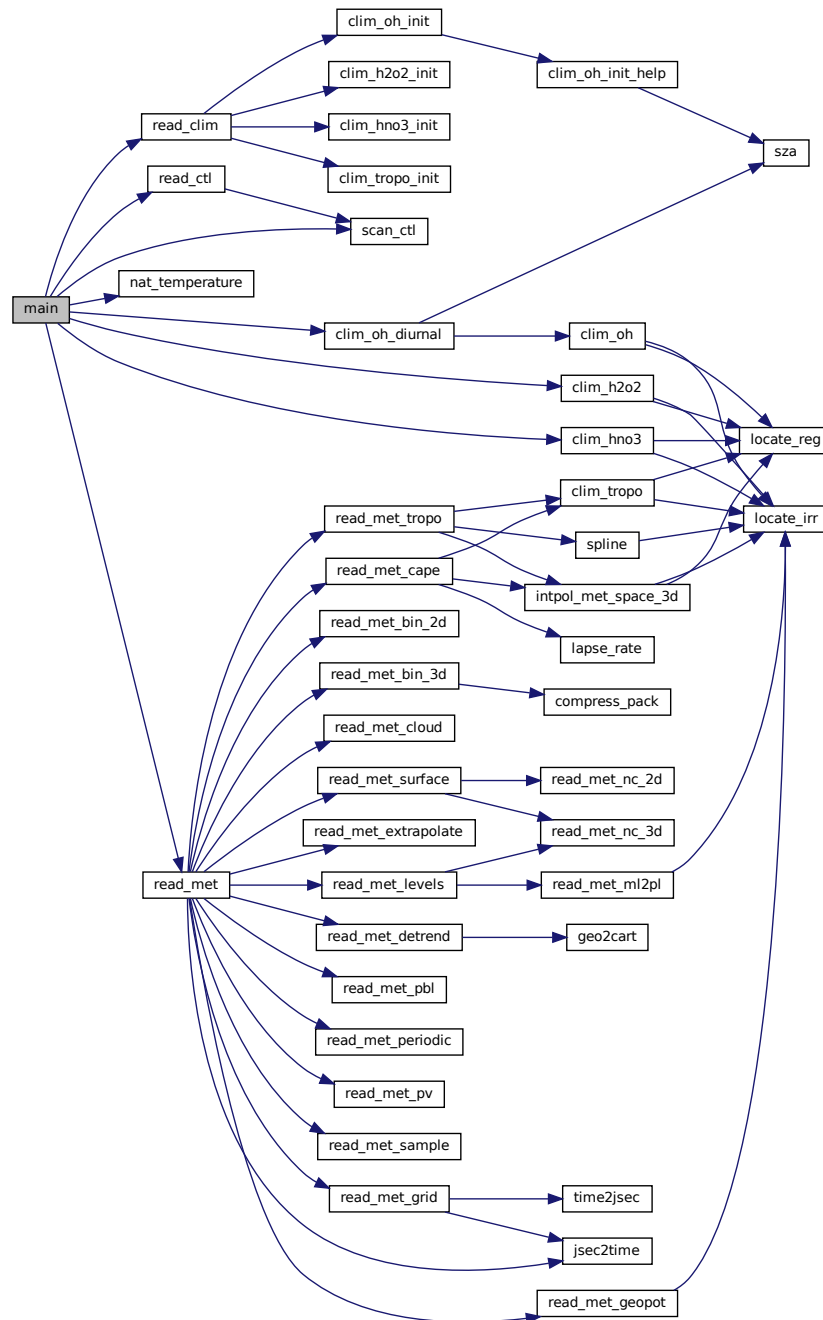
```

```

00207         "# $17 = surface zonal wind [m/s]\n"
00208         "# $18 = surface meridional wind [m/s]\n"
00209         "# $19 = tropopause pressure [hPa]\n"
00210         "# $20 = tropopause geopotential height [km]\n");
00211     fprintf(out,
00212         "# $21 = tropopause temperature [K]\n"
00213         "# $22 = tropopause water vapor [ppv]\n"
00214         "# $23 = cloud liquid water content [kg/kg]\n"
00215         "# $24 = cloud ice water content [kg/kg]\n"
00216         "# $25 = total column cloud water [kg/m^2]\n"
00217         "# $26 = cloud top pressure [hPa]\n"
00218         "# $27 = cloud bottom pressure [hPa]\n"
00219         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00220         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00221         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00222     fprintf(out,
00223         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00224         "# $32 = convective inhibition (CIN) [J/kg]\n"
00225         "# $33 = relative humidity over water [%]\n"
00226         "# $34 = relative humidity over ice [%]\n"
00227         "# $35 = dew point temperature [K]\n"
00228         "# $36 = frost point temperature [K]\n"
00229         "# $37 = NAT temperature [K]\n"
00230         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00231         "# $39 = OH concentration [molec/cm^3]\n"
00232         "# $40 = H2O2 concentration [molec/cm^3]\n");
00233     fprintf(out,
00234         "# $41 = boundary layer pressure [hPa]\n"
00235         "# $42 = number of data points\n"
00236         "# $43 = number of tropopause data points\n"
00237         "# $44 = number of CAPE data points\n\n");
00238
00239     /* Write data... */
00240     for (iz = 0; iz < nz; iz++)
00241     {
00242         fprintf(out,
00243             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n",
00244             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g",
00245             timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00246             latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00247             vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00248             o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00249             psm[iz] / np[iz], tsm[iz] / np[iz], zsm[iz] / np[iz],
00250             usm[iz] / np[iz], vsm[iz] / np[iz], ptm[iz] / npt[iz],
00251             ztm[iz] / npt[iz], tt[iz] / npt[iz], h2otm[iz] / npt[iz],
00252             lwcm[iz] / np[iz], iwcm[iz] / np[iz], clm[iz] / np[iz],
00253             pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
00254             plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
00255             cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
00256             tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
00257             hno3m[iz] / np[iz], ohm[iz] / np[iz], h2o2m[iz] / np[iz],
00258             pblm[iz] / np[iz], np[iz], npt[iz], npc[iz]);
00259     }
00260
00261     /* Close file... */
00262     fclose(out);
00263
00264     /* Free... */
00265     free(clim);
00266     free(met);
00267
00268     return EXIT_SUCCESS;
00269 }

```

Here is the call graph for this function:



5.32 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) 2013-2022 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      clim_t *clim;
00045
00046      met_t *met;
00047
00048      FILE *out;
00049
00050      static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00051          lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00052          wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00053          iwc, iwcm[NZ], ps, psm[NZ], ts, tsm[NZ], zs, zsm[NZ], us, usm[NZ],
00054          vs, vsm[NZ], pbl, pblm[NZ], pt, ptm[NZ], pct, pctm[NZ], pcb, pcbm[NZ],
00055          cl, clm[NZ], plcl, plclm[NZ], plfc, plfcm[NZ], pel, pelm[NZ],
00056          cape, capem[NZ], cin, cinm[NZ], tt, ttm[NZ], zm[NZ], zt, ztm[NZ],
00057          pv, pvm[NZ], plev[NZ], rhm[NZ], rhicem[NZ], tdewm[NZ], ticem[NZ],
00058          tnatm[NZ], hno3m[NZ], ohm[NZ], h2o2m[NZ], cw[3];
00059
00060      static int i, iz, np[NZ], npc[NZ], npt[NZ], nz, ci[3];
00061
00062      /* Allocate... */
00063      ALLOC(clim, clim_t, 1);
00064      ALLOC(met, met_t, 1);
00065
00066      /* Check arguments... */
00067      if (argc < 4)
00068          ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00069
00070      /* Read control parameters... */
00071      read_ctl(argv[1], argc, argv, &ctl);
00072      z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00073      z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00074      dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00075      lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00076      lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00077      dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00078      lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00079      lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00080      dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00081
00082      /* Read climatological data... */
00083      read_clim(&ctl, clim);
00084
00085      /* Loop over input files... */
00086      for (i = 3; i < argc; i++) {
00087
00088          /* Read meteorological data... */
00089          if (!read_met(argv[i], &ctl, clim, met))
00090              continue;
00091
00092          /* Set vertical grid... */
00093          if (z0 < 0)
00094              z0 = Z(met->p[0]);
00095          if (z1 < 0)
00096              z1 = Z(met->p[met->np - 1]);
00097          nz = 0;
00098          if (dz < 0) {
00099              for (iz = 0; iz < met->np; iz++)
00100                  if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00101                      plev[nz] = met->p[iz];
00102                      if (++nz > NZ)
00103                          ERRMSG("Too many pressure levels!");
00103

```

```

00104     }
00105 } else
00106     for (z = z0; z <= z1; z += dz) {
00107     plev[nz] = P(z);
00108     if ((++nz) > NZ)
00109         ERRMSG("Too many pressure levels!");
00110     }
00111
00112 /* Set horizontal grid... */
00113 if (dlon <= 0)
00114     dlon = fabs(met->lon[1] - met->lon[0]);
00115 if (dlat <= 0)
00116     dlat = fabs(met->lat[1] - met->lat[0]);
00117
00118 /* Average... */
00119 for (iz = 0; iz < nz; iz++)
00120     for (lon = lon0; lon <= lon1; lon += dlon)
00121         for (lat = lat0; lat <= lat1; lat += dlat) {
00122
00123             /* Interpolate meteo data... */
00124             INTPOL_SPACE_ALL(plev[iz], lon, lat);
00125
00126             /* Averaging... */
00127             if (gsl_finite(t) && gsl_finite(u)
00128                 && gsl_finite(v) && gsl_finite(w)) {
00129                 timem[iz] += met->time;
00130                 lonm[iz] += lon;
00131                 latm[iz] += lat;
00132                 zm[iz] += z;
00133                 tm[iz] += t;
00134                 um[iz] += u;
00135                 vm[iz] += v;
00136                 wm[iz] += w;
00137                 pvm[iz] += pv;
00138                 h2om[iz] += h2o;
00139                 o3m[iz] += o3;
00140                 lwcm[iz] += lwc;
00141                 iwcm[iz] += iwc;
00142                 psm[iz] += ps;
00143                 tsm[iz] += ts;
00144                 zsm[iz] += zs;
00145                 usm[iz] += us;
00146                 vsm[iz] += vs;
00147                 pblm[iz] += pbl;
00148                 pctm[iz] += pct;
00149                 pcbm[iz] += pcb;
00150                 clm[iz] += cl;
00151                 if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00152                     && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00153                     plclm[iz] += plcl;
00154                     plfcm[iz] += plfc;
00155                     pelm[iz] += pel;
00156                     capem[iz] += cape;
00157                     cinm[iz] += cin;
00158                     npc[iz]++;
00159                 }
00160                 if (gsl_finite(pt)) {
00161                     ptm[iz] += pt;
00162                     ztm[iz] += zt;
00163                     ttm[iz] += tt;
00164                     h2otm[iz] += h2ot;
00165                     npt[iz]++;
00166                 }
00167                 rhm[iz] += RH(plev[iz], t, h2o);
00168                 rhicem[iz] += RHICE(plev[iz], t, h2o);
00169                 tdewm[iz] += TDEW(plev[iz], h2o);
00170                 ticem[iz] += TICE(plev[iz], h2o);
00171                 hno3m[iz] += clim_hno3(clim, met->time, lat, plev[iz]);
00172                 tnatm[iz] +=
00173                     nat_temperature(plev[iz], h2o,
00174                                     clim_hno3(clim, met->time, lat, plev[iz]));
00175                 ohm[iz] +=
00176                     clim_oh_diurnal(&ctl, clim, met->time, plev[iz], lon, lat);
00177                 h2o2m[iz] += clim_h2o2(clim, met->time, lat, plev[iz]);
00178                 np[iz]++;
00179             }
00180         }
00181 }
00182
00183 /* Create output file... */
00184 LOG(1, "Write meteorological data file: %s", argv[2]);
00185 if (!(out = fopen(argv[2], "w")))
00186     ERRMSG("Cannot create file!");
00187
00188 /* Write header... */
00189 fprintf(out,
00190     "# $1 = time [s]\n"

```

```

00191         "# $2 = altitude [km]\n"
00192         "# $3 = longitude [deg]\n"
00193         "# $4 = latitude [deg]\n"
00194         "# $5 = pressure [hPa]\n"
00195         "# $6 = temperature [K]\n"
00196         "# $7 = zonal wind [m/s]\n"
00197         "# $8 = meridional wind [m/s]\n"
00198         "# $9 = vertical velocity [hPa/s]\n"
00199         "# $10 = H2O volume mixing ratio [ppv]\n");
00200     fprintf(out,
00201         "# $11 = O3 volume mixing ratio [ppv]\n"
00202         "# $12 = geopotential height [km]\n"
00203         "# $13 = potential vorticity [PVU]\n"
00204         "# $14 = surface pressure [hPa]\n"
00205         "# $15 = surface temperature [K]\n"
00206         "# $16 = surface geopotential height [km]\n"
00207         "# $17 = surface zonal wind [m/s]\n"
00208         "# $18 = surface meridional wind [m/s]\n"
00209         "# $19 = tropopause pressure [hPa]\n"
00210         "# $20 = tropopause geopotential height [km]\n");
00211     fprintf(out,
00212         "# $21 = tropopause temperature [K]\n"
00213         "# $22 = tropopause water vapor [ppv]\n"
00214         "# $23 = cloud liquid water content [kg/kg]\n"
00215         "# $24 = cloud ice water content [kg/kg]\n"
00216         "# $25 = total column cloud water [kg/m^2]\n"
00217         "# $26 = cloud top pressure [hPa]\n"
00218         "# $27 = cloud bottom pressure [hPa]\n"
00219         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00220         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00221         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00222     fprintf(out,
00223         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00224         "# $32 = convective inhibition (CIN) [J/kg]\n"
00225         "# $33 = relative humidity over water [%]\n"
00226         "# $34 = relative humidity over ice [%]\n"
00227         "# $35 = dew point temperature [K]\n"
00228         "# $36 = frost point temperature [K]\n"
00229         "# $37 = NAT temperature [K]\n"
00230         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00231         "# $39 = OH concentration [molec/cm^3]\n"
00232         "# $40 = H2O2 concentration [molec/cm^3]\n");
00233     fprintf(out,
00234         "# $41 = boundary layer pressure [hPa]\n"
00235         "# $42 = number of data points\n"
00236         "# $43 = number of tropopause data points\n"
00237         "# $44 = number of CAPE data points\n\n");
00238
00239     /* Write data... */
00240     for (iz = 0; iz < nz; iz++)
00241         fprintf(out,
00242             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00243             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00244             timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00245             latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00246             vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00247             o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00248             psm[iz] / np[iz], tsm[iz] / np[iz], zsm[iz] / np[iz],
00249             usm[iz] / np[iz], vsm[iz] / np[iz], ptm[iz] / npt[iz],
00250             ztm[iz] / npt[iz], ttm[iz] / npt[iz], h2otm[iz] / npt[iz],
00251             lwcm[iz] / np[iz], iwcm[iz] / np[iz], clm[iz] / np[iz],
00252             pctm[iz] / np[iz], pcbm[iz] / np[iz], plclm[iz] / npc[iz],
00253             plfcm[iz] / npc[iz], pelm[iz] / npc[iz], capem[iz] / npc[iz],
00254             cinm[iz] / npc[iz], rhm[iz] / np[iz], rhicem[iz] / np[iz],
00255             tdewm[iz] / np[iz], ticem[iz] / np[iz], tnatm[iz] / np[iz],
00256             hno3m[iz] / np[iz], ohm[iz] / np[iz], h2o2m[iz] / np[iz],
00257             pblm[iz] / np[iz], np[iz], npt[iz], npc[iz]);
00258
00259     /* Close file... */
00260     fclose(out);
00261
00262     /* Free... */
00263     free(clim);
00264     free(met);
00265
00266     return EXIT_SUCCESS;
00267 }
00268

```

5.33 met_sample.c File Reference

Sample meteorological data at given geolocations.

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

Functions

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

5.33.1 Detailed Description

Sample meteorological data at given geolocations.

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

5.33.2 Function Documentation

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

Definition at line 31 of file [met_sample.c](#).

```
00033     {
00034
00035     ctl_t ctl;
00036
00037     clim_t *clim;
00038
00039     atm_t *atm;
00040
00041     met_t *met0, *met1;
00042
00043     FILE *out;
00044
00045     double h2o, h2ot, o3, lwc, iwc, p0, pl, ps, ts, zs, us, vs, pbl, pt,
00046            pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00047            zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00048
00049     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00050
00051     /* Check arguments... */
00052     if (argc < 3)
00053         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00054
00055     /* Allocate... */
00056     ALLOC(clim, clim_t, 1);
00057     ALLOC(atm, atm_t, 1);
00058     ALLOC(met0, met_t, 1);
00059     ALLOC(met1, met_t, 1);
00060
00061     /* Read control parameters... */
00062     read_ctl(argv[1], argc, argv, &ctl);
00063     geopot =
00064         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00065     grid_time =
00066         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00067     grid_z =
00068         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00069     grid_lon =
00070         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00071     grid_lat =
00072         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00073
00074     /* Read climatological data... */
00075     read_clim(&ctl, clim);
00076
00077     /* Read atmospheric data... */
00078     if (!read_atm(argv[3], &ctl, atm))
```

```

00079     ERRMSG("Cannot open file!");
00080
00081     /* Create output file... */
00082     LOG(1, "Write meteorological data file: %s", argv[2]);
00083     if (! (out = fopen(argv[2], "w")))
00084         ERRMSG("Cannot create file!");
00085
00086     /* Write header... */
00087     fprintf(out,
00088         "# $1 = time [s]\n"
00089         "# $2 = altitude [km]\n"
00090         "# $3 = longitude [deg]\n"
00091         "# $4 = latitude [deg]\n"
00092         "# $5 = pressure [hPa]\n"
00093         "# $6 = temperature [K]\n"
00094         "# $7 = zonal wind [m/s]\n"
00095         "# $8 = meridional wind [m/s]\n"
00096         "# $9 = vertical velocity [hPa/s]\n"
00097         "# $10 = H2O volume mixing ratio [ppv]\n");
00098     fprintf(out,
00099         "# $11 = O3 volume mixing ratio [ppv]\n"
00100         "# $12 = geopotential height [km]\n"
00101         "# $13 = potential vorticity [PVU]\n"
00102         "# $14 = surface pressure [hPa]\n"
00103         "# $15 = surface temperature [K]\n"
00104         "# $16 = surface geopotential height [km]\n"
00105         "# $17 = surface zonal wind [m/s]\n"
00106         "# $18 = surface meridional wind [m/s]\n"
00107         "# $19 = tropopause pressure [hPa]\n"
00108         "# $20 = tropopause geopotential height [km]\n");
00109     fprintf(out,
00110         "# $21 = tropopause temperature [K]\n"
00111         "# $22 = tropopause water vapor [ppv]\n"
00112         "# $23 = cloud liquid water content [kg/kg]\n"
00113         "# $24 = cloud ice water content [kg/kg]\n"
00114         "# $25 = total column cloud water [kg/m^2]\n"
00115         "# $26 = cloud top pressure [hPa]\n"
00116         "# $27 = cloud bottom pressure [hPa]\n"
00117         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00118         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00119         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00120     fprintf(out,
00121         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00122         "# $32 = convective inhibition (CIN) [J/kg]\n"
00123         "# $33 = relative humidity over water [%]\n"
00124         "# $34 = relative humidity over ice [%]\n"
00125         "# $35 = dew point temperature [K]\n"
00126         "# $36 = frost point temperature [K]\n"
00127         "# $37 = NAT temperature [K]\n"
00128         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00129         "# $39 = OH concentration [molec/cm^3]\n"
00130         "# $40 = H2O2 concentration [molec/cm^3]\n"
00131         "# $41 = boundary layer pressure [hPa]\n");
00132
00133     /* Loop over air parcels... */
00134     for (ip = 0; ip < atm->np; ip++) {
00135
00136         /* Get meteorological data... */
00137         get_met(&ctl, clim, atm->time[ip], &met0, &met1);
00138
00139         /* Set reference pressure for interpolation... */
00140         double pref = atm->p[ip];
00141         if (geopot) {
00142             zref = Z(pref);
00143             p0 = met0->p[0];
00144             p1 = met0->p[met0->np - 1];
00145             for (it = 0; it < 24; it++) {
00146                 pref = 0.5 * (p0 + p1);
00147                 intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
00148                                     atm->lon[ip], atm->lat[ip], &zm, ci, cw, l);
00149                 if (zref > zm || !gsl_finite(zm))
00150                     p0 = pref;
00151                 else
00152                     p1 = pref;
00153             }
00154             pref = 0.5 * (p0 + p1);
00155         }
00156
00157         /* Interpolate meteo data... */
00158         INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
00159
00160         /* Make blank lines... */
00161         if (ip == 0 || (grid_time && atm->time[ip] != time_old)
00162             || (grid_z && atm->p[ip] != p_old)
00163             || (grid_lon && atm->lon[ip] != lon_old)
00164             || (grid_lat && atm->lat[ip] != lat_old))
00165             fprintf(out, "\n");

```

```

00166     time_old = atm->time[ip];
00167     p_old = atm->p[ip];
00168     lon_old = atm->lon[ip];
00169     lat_old = atm->lat[ip];
00170
00171     /* Write data... */
00172     fprintf(out,
00173         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00174         " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00175         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00176         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
00177         pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
00178         cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
00179         TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
00180         nat_temperature(atm->p[ip], h2o,
00181             clim_hno3(clim, atm->time[ip], atm->lat[ip],
00182                 atm->p[ip])), clim_hno3(clim,
00183                 atm->time[ip],
00184                 atm->lat[ip],
00185                 atm->p[ip]),
00186         clim_oh_diurnal(&ctl, clim, atm->time[ip], atm->p[ip],
00187             atm->lon[ip], atm->lat[ip]),
00188         clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
00189 }
00190
00191 /* Close file... */
00192 fclose(out);
00193
00194 /* Free... */
00195 free(clim);
00196 free(atm);
00197 free(met0);
00198 free(met1);
00199
00200 return EXIT_SUCCESS;
00201 }

```



```

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     clim_t *clim;
00038
00039     atm_t *atm;
00040
00041     met_t *met0, *met1;
00042
00043     FILE *out;
00044
00045     double h2o, h2ot, o3, lwc, iwc, p0, pl, ps, ts, zs, us, vs, pbl, pt,
00046         pct, pcb, cl, plcl, plfc, pel, cape, cin, pv, t, tt, u, v, w, z, zm, zref,
00047         zt, cw[3], time_old = -999, p_old = -999, lon_old = -999, lat_old = -999;
00048
00049     int geopot, grid_time, grid_z, grid_lon, grid_lat, ip, it, ci[3];
00050
00051     /* Check arguments... */
00052     if (argc < 3)
00053         ERRMSG("Give parameters: <ctl> <sample.tab> <atm_in>");
00054
00055     /* Allocate... */
00056     ALLOC(clim, clim_t, 1);
00057     ALLOC(atm, atm_t, 1);
00058     ALLOC(met0, met_t, 1);
00059     ALLOC(met1, met_t, 1);
00060
00061     /* Read control parameters... */
00062     read_ctl(argv[1], argc, argv, &ctl);
00063     geopot =
00064         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00065     grid_time =
00066         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_TIME", -1, "0", NULL);
00067     grid_z =
00068         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_Z", -1, "0", NULL);
00069     grid_lon =
00070         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LON", -1, "0", NULL);
00071     grid_lat =
00072         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GRID_LAT", -1, "0", NULL);
00073
00074     /* Read climatological data... */
00075     read_clim(&ctl, clim);
00076
00077     /* Read atmospheric data... */
00078     if (!read_atm(argv[3], &ctl, atm))
00079         ERRMSG("Cannot open file!");
00080
00081     /* Create output file... */
00082     LOG(1, "Write meteorological data file: %s", argv[2]);
00083     if (!(out = fopen(argv[2], "w")))
00084         ERRMSG("Cannot create file!");
00085
00086     /* Write header... */
00087     fprintf(out,
00088         "# $1 = time [s]\n"
00089         "# $2 = altitude [km]\n"
00090         "# $3 = longitude [deg]\n"
00091         "# $4 = latitude [deg]\n"
00092         "# $5 = pressure [hPa]\n"
00093         "# $6 = temperature [K]\n"
00094         "# $7 = zonal wind [m/s]\n"
00095         "# $8 = meridional wind [m/s]\n"
00096         "# $9 = vertical velocity [hPa/s]\n"
00097         "# $10 = H2O volume mixing ratio [ppv]\n");
00098     fprintf(out,
00099         "# $11 = O3 volume mixing ratio [ppv]\n"
00100         "# $12 = geopotential height [km]\n"
00101         "# $13 = potential vorticity [PVU]\n"
00102         "# $14 = surface pressure [hPa]\n"
00103         "# $15 = surface temperature [K]\n"
00104         "# $16 = surface geopotential height [km]\n"
00105         "# $17 = surface zonal wind [m/s]\n"
00106         "# $18 = surface meridional wind [m/s]\n"
00107         "# $19 = tropopause pressure [hPa]\n"
00108         "# $20 = tropopause geopotential height [km]\n");
00109     fprintf(out,
00110         "# $21 = tropopause temperature [K]\n"

```



```

001111     "# $22 = tropopause water vapor [ppv]\n"
001112     "# $23 = cloud liquid water content [kg/kg]\n"
001113     "# $24 = cloud ice water content [kg/kg]\n"
001114     "# $25 = total column cloud water [kg/m^2]\n"
001115     "# $26 = cloud top pressure [hPa]\n"
001116     "# $27 = cloud bottom pressure [hPa]\n"
001117     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
001118     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
001119     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
001120 fprintf(out,
001121     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
001122     "# $32 = convective inhibition (CIN) [J/kg]\n"
001123     "# $33 = relative humidity over water [%]\n"
001124     "# $34 = relative humidity over ice [%]\n"
001125     "# $35 = dew point temperature [K]\n"
001126     "# $36 = frost point temperature [K]\n"
001127     "# $37 = NAT temperature [K]\n"
001128     "# $38 = HNO3 volume mixing ratio [ppv]\n"
001129     "# $39 = OH concentration [molec/cm^3]\n"
001130     "# $40 = H2O2 concentration [molec/cm^3]\n"
001131     "# $41 = boundary layer pressure [hPa]\n");
001132
001133 /* Loop over air parcels... */
001134 for (ip = 0; ip < atm->np; ip++) {
001135
001136     /* Get meteorological data... */
001137     get_met(&ctl, clim, atm->time[ip], &met0, &met1);
001138
001139     /* Set reference pressure for interpolation... */
001140     double pref = atm->p[ip];
001141     if (geopot) {
001142         zref = Z(pref);
001143         p0 = met0->p[0];
001144         p1 = met0->p[met0->np - 1];
001145         for (it = 0; it < 24; it++) {
001146             pref = 0.5 * (p0 + p1);
001147             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->time[ip], pref,
001148                               atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
001149             if (zref > zm || !gs1_finite(zm))
001150                 p0 = pref;
001151             else
001152                 p1 = pref;
001153         }
001154         pref = 0.5 * (p0 + p1);
001155     }
001156
001157     /* Interpolate meteo data... */
001158     INTPOL_TIME_ALL(atm->time[ip], pref, atm->lon[ip], atm->lat[ip]);
001159
001160     /* Make blank lines... */
001161     if (ip == 0 || (grid_time && atm->time[ip] != time_old)
001162         || (grid_z && atm->p[ip] != p_old)
001163         || (grid_lon && atm->lon[ip] != lon_old)
001164         || (grid_lat && atm->lat[ip] != lat_old))
001165         fprintf(out, "\n");
001166     time_old = atm->time[ip];
001167     p_old = atm->p[ip];
001168     lon_old = atm->lon[ip];
001169     lat_old = atm->lat[ip];
001170
001171     /* Write data... */
001172     fprintf(out,
001173         "%2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
001174         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
001175         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, ts, zs, us, vs,
001176         pt, zt, tt, h2ot, lwc, iwc, cl, pct, pcb, plcl, plfc, pel, cape,
001177         cin, RH(atm->p[ip], t, h2o), RHICE(atm->p[ip], t, h2o),
001178         TDEW(atm->p[ip], h2o), TICE(atm->p[ip], h2o),
001179         nat_temperature(atm->p[ip], h2o,
001180             clim_hno3(clim, atm->time[ip], atm->lat[ip],
001181                       atm->p[ip])), clim_hno3(clim,
001182                                               atm->time[ip],
001183                                               atm->lat[ip],
001184                                               atm->p[ip]),
001185         clim_oh_diurnal(&ctl, clim, atm->time[ip], atm->p[ip],
001186                       atm->lon[ip], atm->lat[ip]),
001187         clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip]), pbl);
001188 }
001189
001190 /* Close file... */
001191 fclose(out);
001192
001193 /* Free... */
001194 free(clim);
001195 free(atm);
001196 free(met0);

```

```
00198     free(met1);
00199
00200     return EXIT_SUCCESS;
00201 }
```

5.35 met_spec.c File Reference

Spectral analysis of meteorological data.

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

Macros

- `#define PMAX EX`
Maximum number of data points for spectral analysis.

Functions

- void [fft_help](#) (double *fcReal, double *fclmag, int n)
- int [main](#) (int argc, char *argv[])

5.35.1 Detailed Description

Spectral analysis of meteorological data.

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

5.35.2 Macro Definition Documentation

5.35.2.1 PMAX `#define PMAX EX`

Maximum number of data points for spectral analysis.

Definition at line [32](#) of file [met_spec.c](#).

5.35.3 Function Documentation

5.35.3.1 `fft_help()` `void fft_help (`
`double * fcReal,`
`double * fcImag,`
`int n)`

Definition at line 150 of file `met_spec.c`.

```
00153     {
00154
00155     gsl_fft_complex_wavetable *wavetable;
00156     gsl_fft_complex_workspace *workspace;
00157
00158     double data[2 * PMAX];
00159
00160     int i;
00161
00162     /* Check size... */
00163     if (n > PMAX)
00164         ERRMSG("Too many data points!");
00165
00166     /* Allocate... */
00167     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00168     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00169
00170     /* Set data (real, complex)... */
00171     for (i = 0; i < n; i++) {
00172         data[2 * i] = fcReal[i];
00173         data[2 * i + 1] = fcImag[i];
00174     }
00175
00176     /* Calculate FFT... */
00177     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00178
00179     /* Copy data... */
00180     for (i = 0; i < n; i++) {
00181         fcReal[i] = data[2 * i];
00182         fcImag[i] = data[2 * i + 1];
00183     }
00184
00185     /* Free... */
00186     gsl_fft_complex_wavetable_free(wavetable);
00187     gsl_fft_complex_workspace_free(workspace);
00188 }
```

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

Definition at line 47 of file `met_spec.c`.

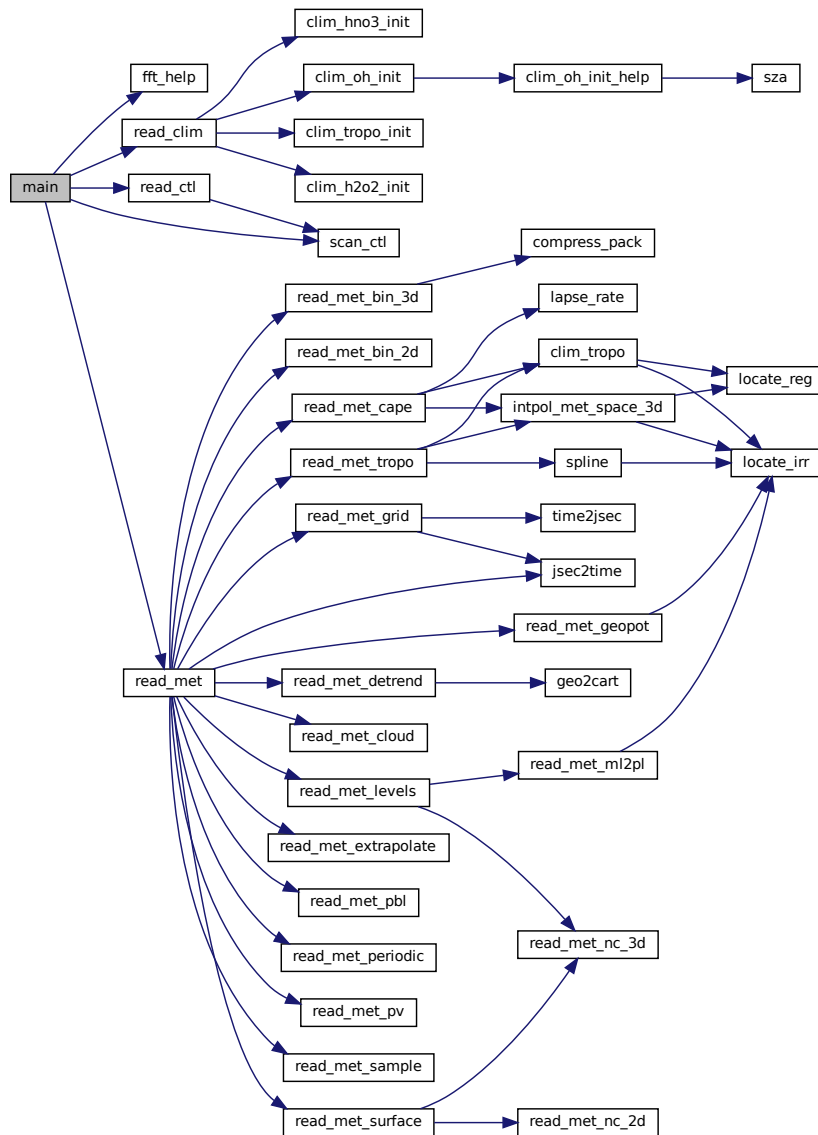
```
00049     {
00050
00051     ctl_t ctl;
00052
00053     clim_t *clim;
00054
00055     met_t *met;
00056
00057     FILE *out;
00058
00059     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00060         wavemax;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     wavemax =
00073         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00074
00075     /* Read climatological data... */
00076     read_clim(&ctl, clim);
00077 }
```

```

00078  /* Read meteorological data... */
00079  if (!read_met(argv[3], &ctl, clim, met))
00080      ERRMSG("Cannot read meteo data!");
00081
00082  /* Create output file... */
00083  LOG(1, "Write spectral data file: %s", argv[2]);
00084  if (!(out = fopen(argv[2], "w")))
00085      ERRMSG("Cannot create file!");
00086
00087  /* Write header... */
00088  fprintf(out,
00089      "# $1 = time [s]\n"
00090      "# $2 = altitude [km]\n"
00091      "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00092  for (int ix = 0; ix <= wavemax; ix++) {
00093      fprintf(out, "# $%d = wavelength (PW%d) [km]\n", 5 + 3 * ix, ix);
00094      fprintf(out, "# $%d = amplitude (PW%d) [K]\n", 6 + 3 * ix, ix);
00095      fprintf(out, "# $%d = phase (PW%d) [deg]\n", 7 + 3 * ix, ix);
00096  }
00097
00098  /* Loop over pressure levels... */
00099  for (int ip = 0; ip < met->np; ip++) {
00100
00101      /* Write output... */
00102      fprintf(out, "\n");
00103
00104      /* Loop over latitudes... */
00105      for (int iy = 0; iy < met->ny; iy++) {
00106
00107          /* Copy data... */
00108          for (int ix = 0; ix < met->nx; ix++) {
00109              cutReal[ix] = met->t[ix][iy][ip];
00110              cutImag[ix] = 0.0;
00111          }
00112
00113          /* FFT... */
00114          fft_help(cutReal, cutImag, met->nx);
00115
00116          /*
00117           * Get wavelength, amplitude, and phase:
00118           *  $A(x) = A[0] + A[1] * \cos(2 \pi x / lx[1] + \phi[1]) + A[2] * \cos...$ 
00119           */
00120          for (int ix = 0; ix < met->nx; ix++) {
00121              lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00122                  / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00123              A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00124                  * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00125              phi[ix]
00126                  = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00127          }
00128
00129          /* Write data... */
00130          fprintf(out, "%.2f %g %g", met->time, Z(met->p[ip]), 0.0,
00131              met->lat[iy]);
00132          for (int ix = 0; ix <= wavemax; ix++)
00133              fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00134          fprintf(out, "\n");
00135      }
00136  }
00137
00138  /* Close file... */
00139  fclose(out);
00140
00141  /* Free... */
00142  free(clim);
00143  free(met);
00144
00145  return EXIT_SUCCESS;
00146 }

```

Here is the call graph for this function:



5.36 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     clim_t *clim;
00054
00055     met_t *met;
00056
00057     FILE *out;
00058
00059     static double cutImag[PMAX], cutReal[PMAX], lx[PMAX], A[PMAX], phi[PMAX],
00060         wavemax;
00061
00062     /* Allocate... */
00063     ALLOC(clim, clim_t, 1);
00064     ALLOC(met, met_t, 1);
00065
00066     /* Check arguments... */
00067     if (argc < 4)
00068         ERRMSG("Give parameters: <ctl> <spec.tab> <met0>");
00069
00070     /* Read control parameters... */
00071     read_ctl(argv[1], argc, argv, &ctl);
00072     wavemax =
00073         (int) scan_ctl(argv[1], argc, argv, "SPEC_WAVEMAX", -1, "7", NULL);
00074
00075     /* Read climatological data... */
00076     read_clim(&ctl, clim);
00077
00078     /* Read meteorological data... */
00079     if (!read_met(argv[3], &ctl, clim, met))
00080         ERRMSG("Cannot read meteo data!");
00081
00082     /* Create output file... */
00083     LOG(1, "Write spectral data file: %s", argv[2]);
00084     if (!(out = fopen(argv[2], "w")))
00085         ERRMSG("Cannot create file!");
00086
00087     /* Write header... */
00088     fprintf(out,
00089         "# $1 = time [s]\n"
00090         "# $2 = altitude [km]\n"
00091         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00092     for (int ix = 0; ix <= wavemax; ix++) {
00093         fprintf(out, "# $d = wavelength (PW$d) [km]\n", 5 + 3 * ix, ix);
00094         fprintf(out, "# $d = amplitude (PW$d) [K]\n", 6 + 3 * ix, ix);
00095         fprintf(out, "# $d = phase (PW$d) [deg]\n", 7 + 3 * ix, ix);
00096     }
00097
00098     /* Loop over pressure levels... */
00099     for (int ip = 0; ip < met->np; ip++) {
00100
00101         /* Write output... */
00102         fprintf(out, "\n");
00103
00104         /* Loop over latitudes... */
00105         for (int iy = 0; iy < met->ny; iy++) {
00106
00107             /* Copy data... */
00108             for (int ix = 0; ix < met->nx; ix++) {
00109                 cutReal[ix] = met->t[ix][iy][ip];
00110                 cutImag[ix] = 0.0;
00111             }
00112

```

```

00113      /* FFT... */
00114      fft_help(cutReal, cutImag, met->nx);
00115
00116      /*
00117       * Get wavelength, amplitude, and phase:
00118       *  $A(x) = A[0] + A[1] * \cos(2 \pi x / lx[1] + \phi[1]) + A[2] * \cos...$ 
00119       */
00120      for (int ix = 0; ix < met->nx; ix++) {
00121          lx[ix] = DEG2DX(met->lon[met->nx - 1] - met->lon[0], met->lat[iy])
00122              / ((ix < met->nx / 2) ? (double) ix : -(double) (met->nx - ix));
00123          A[ix] = (ix == 0 ? 1.0 : 2.0) / (met->nx)
00124              * sqrt(gsl_pow_2(cutReal[ix]) + gsl_pow_2(cutImag[ix]));
00125          phi[ix]
00126              = 180. / M_PI * atan2(cutImag[ix], cutReal[ix]);
00127      }
00128
00129      /* Write data... */
00130      fprintf(out, "%.2f %g %g %g", met->time, Z(met->p[ip]), 0.0,
00131          met->lat[iy]);
00132      for (int ix = 0; ix <= wavemax; ix++)
00133          fprintf(out, " %g %g %g", lx[ix], A[ix], phi[ix]);
00134      fprintf(out, "\n");
00135  }
00136  }
00137
00138  /* Close file... */
00139  fclose(out);
00140
00141  /* Free... */
00142  free(clim);
00143  free(met);
00144
00145  return EXIT_SUCCESS;
00146 }
00147
00148 /*****
00149
00150 void fft_help(
00151     double *fcReal,
00152     double *fcImag,
00153     int n) {
00154
00155     gsl_fft_complex_wavetable *wavetable;
00156     gsl_fft_complex_workspace *workspace;
00157
00158     double data[2 * PMAX];
00159
00160     int i;
00161
00162     /* Check size... */
00163     if (n > PMAX)
00164         ERRMSG("Too many data points!");
00165
00166     /* Allocate... */
00167     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00168     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00169
00170     /* Set data (real, complex)... */
00171     for (i = 0; i < n; i++) {
00172         data[2 * i] = fcReal[i];
00173         data[2 * i + 1] = fcImag[i];
00174     }
00175
00176     /* Calculate FFT... */
00177     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00178
00179     /* Copy data... */
00180     for (i = 0; i < n; i++) {
00181         fcReal[i] = data[2 * i];
00182         fcImag[i] = data[2 * i + 1];
00183     }
00184
00185     /* Free... */
00186     gsl_fft_complex_wavetable_free(wavetable);
00187     gsl_fft_complex_workspace_free(workspace);
00188 }

```

5.37 met_subgrid.c File Reference

Calculate standard deviations of horizontal wind and vertical velocity.

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

Functions

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

5.37.1 Detailed Description

Calculate standard deviations of horizontal wind and vertical velocity.

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

5.37.2 Function Documentation

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

Definition at line 31 of file [met_subgrid.c](#).

```
00033     {
00034
00035     ctl_t ctl;
00036
00037     clim_t *clim;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];
00044
00045     static float u[16], v[16], w[16];
00046
00047     static int i, ix, iy, iz, n[EP][EY];
00048
00049     /* Allocate... */
00050     ALLOC(clim, clim_t, 1);
00051     ALLOC(met0, met_t, 1);
00052     ALLOC(met1, met_t, 1);
00053
00054     /* Check arguments... */
00055     if (argc < 4 && argc % 2 != 0)
00056         ERRMSG
00057             ("Give parameters: <ctl> <subgrid.tab> <met0> <met1> [ <met0> <met1> ... ]");
00058
00059     /* Read control parameters... */
00060     read_ctl(argv[1], argc, argv, &ctl);
00061
00062     /* Read climatological data... */
00063     read_clim(&ctl, clim);
00064
00065     /* Loop over data files... */
00066     for (i = 3; i < argc - 1; i += 2) {
00067
00068         /* Read meteorological data... */
00069         if (!read_met(argv[i], &ctl, clim, met0))
00070             ERRMSG("Cannot open file!");
00071         if (!read_met(argv[i + 1], &ctl, clim, met1))
00072             ERRMSG("Cannot open file!");
00073
00074         /* Loop over grid boxes... */
00075         for (ix = 0; ix < met0->nx - 1; ix++)
00076             for (iy = 0; iy < met0->ny - 1; iy++)
00077                 for (iz = 0; iz < met0->np - 1; iz++) {
00078
00079                     /* Collect local wind data... */
00080                     u[0] = met0->u[ix][iy][iz];
00081                     u[1] = met0->u[ix + 1][iy][iz];
00082                     u[2] = met0->u[ix][iy + 1][iz];
00083                     u[3] = met0->u[ix + 1][iy + 1][iz];
00084                     u[4] = met0->u[ix][iy][iz + 1];
```



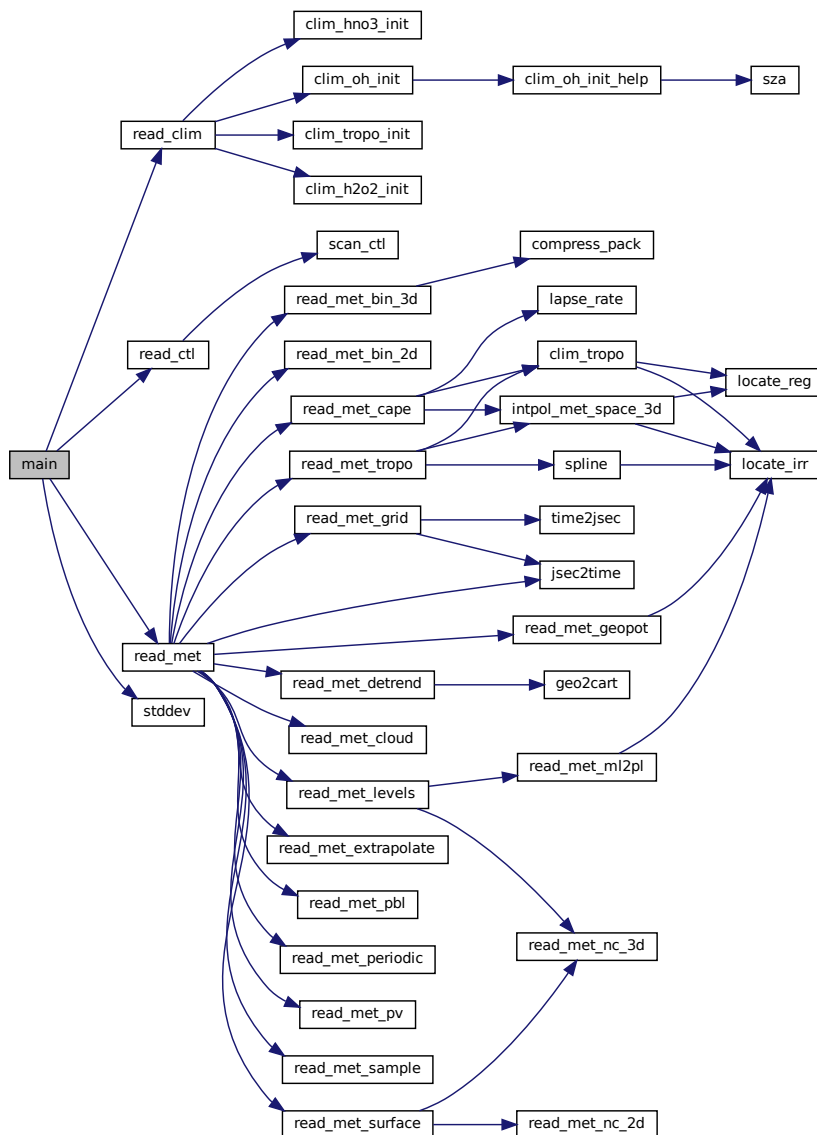
```

00085     u[5] = met0->u[ix + 1][iy][iz + 1];
00086     u[6] = met0->u[ix][iy + 1][iz + 1];
00087     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00088
00089     v[0] = met0->v[ix][iy][iz];
00090     v[1] = met0->v[ix + 1][iy][iz];
00091     v[2] = met0->v[ix][iy + 1][iz];
00092     v[3] = met0->v[ix + 1][iy + 1][iz];
00093     v[4] = met0->v[ix][iy][iz + 1];
00094     v[5] = met0->v[ix + 1][iy][iz + 1];
00095     v[6] = met0->v[ix][iy + 1][iz + 1];
00096     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00097
00098     w[0] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
00099     w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
00100     w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
00101     w[3] =
00102         (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
00103     w[4] =
00104         (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
00105     w[5] =
00106         (float) (1e3 *
00107             DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
00108     w[6] =
00109         (float) (1e3 *
00110             DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
00111     w[7] =
00112         (float) (1e3 *
00113             DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[iz + 1]));
00114
00115     /* Collect local wind data... */
00116     u[8] = met1->u[ix][iy][iz];
00117     u[9] = met1->u[ix + 1][iy][iz];
00118     u[10] = met1->u[ix][iy + 1][iz];
00119     u[11] = met1->u[ix + 1][iy + 1][iz];
00120     u[12] = met1->u[ix][iy][iz + 1];
00121     u[13] = met1->u[ix + 1][iy][iz + 1];
00122     u[14] = met1->u[ix][iy + 1][iz + 1];
00123     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00124
00125     v[8] = met1->v[ix][iy][iz];
00126     v[9] = met1->v[ix + 1][iy][iz];
00127     v[10] = met1->v[ix][iy + 1][iz];
00128     v[11] = met1->v[ix + 1][iy + 1][iz];
00129     v[12] = met1->v[ix][iy][iz + 1];
00130     v[13] = met1->v[ix + 1][iy][iz + 1];
00131     v[14] = met1->v[ix][iy + 1][iz + 1];
00132     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00133
00134     w[8] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
00135     w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
00136     w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
00137     w[11] =
00138         (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
00139     w[12] =
00140         (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
00141     w[13] =
00142         (float) (1e3 *
00143             DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
00144     w[14] =
00145         (float) (1e3 *
00146             DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
00147     w[15] =
00148         (float) (1e3 *
00149             DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));
00150
00151     /* Get standard deviations of local wind data... */
00152     usig[iz][iy] += stddev(u, 16);
00153     vsig[iz][iy] += stddev(v, 16);
00154     wsig[iz][iy] += stddev(w, 16);
00155     n[iz][iy]++;
00156
00157     /* Check surface pressure... */
00158     if (met0->p[iz] > met0->ps[ix][iy]
00159         || met1->p[iz] > met1->ps[ix][iy]) {
00160         usig[iz][iy] = GSL_NAN;
00161         vsig[iz][iy] = GSL_NAN;
00162         wsig[iz][iy] = GSL_NAN;
00163         n[iz][iy] = 0;
00164     }
00165 }
00166 }
00167
00168 /* Create output file... */
00169 LOG(1, "Write subgrid data file: %s", argv[2]);
00170 if (!out = fopen(argv[2], "w"))
00171     ERRMSG("Cannot create file!");

```

```
00172
00173 /* Write header... */
00174 fprintf(out,
00175     "# $1 = time [s]\n"
00176     "# $2 = altitude [km]\n"
00177     "# $3 = longitude [deg]\n"
00178     "# $4 = latitude [deg]\n"
00179     "# $5 = zonal wind standard deviation [m/s]\n"
00180     "# $6 = meridional wind standard deviation [m/s]\n"
00181     "# $7 = vertical velocity standard deviation [m/s]\n"
00182     "# $8 = number of data points\n");
00183
00184 /* Write output... */
00185 for (iy = 0; iy < met0->ny - 1; iy++) {
00186     fprintf(out, "\n");
00187     for (iz = 0; iz < met0->np - 1; iz++)
00188         fprintf(out, "%.2f %g %g %g %g %g %d\n",
00189             0.5 * (met0->time + met1->time),
00190             0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
00191             0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
00192             usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
00193             wsig[iz][iy] / n[iz][iy], n[iz][iy]);
00194 }
00195
00196 /* Close file... */
00197 fclose(out);
00198
00199 /* Free... */
00200 free(clim);
00201 free(met0);
00202 free(met1);
00203
00204 return EXIT_SUCCESS;
00205 }
```

Here is the call graph for this function:



5.38 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     clim_t *clim;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     static double usig[EP][EY], vsig[EP][EY], wsig[EP][EY];
00044
00045     static float u[16], v[16], w[16];
00046
00047     static int i, ix, iy, iz, n[EP][EY];
00048
00049     /* Allocate... */
00050     ALLOC(clim, clim_t, 1);
00051     ALLOC(met0, met_t, 1);
00052     ALLOC(met1, met_t, 1);
00053
00054     /* Check arguments... */
00055     if (argc < 4 && argc % 2 != 0)
00056         ERRMSG
00057             ("Give parameters: <ctl> <subgrid.tab> <met0> <met1> [ <met0> <met1> ... ]");
00058
00059     /* Read control parameters... */
00060     read_ctl(argv[1], argc, argv, &ctl);
00061
00062     /* Read climatological data... */
00063     read_clim(&ctl, clim);
00064
00065     /* Loop over data files... */
00066     for (i = 3; i < argc - 1; i += 2) {
00067
00068         /* Read meteorological data... */
00069         if (!read_met(argv[i], &ctl, clim, met0))
00070             ERRMSG("Cannot open file!");
00071         if (!read_met(argv[i + 1], &ctl, clim, met1))
00072             ERRMSG("Cannot open file!");
00073
00074         /* Loop over grid boxes... */
00075         for (ix = 0; ix < met0->nx - 1; ix++)
00076             for (iy = 0; iy < met0->ny - 1; iy++)
00077                 for (iz = 0; iz < met0->np - 1; iz++) {
00078
00079                     /* Collect local wind data... */
00080                     u[0] = met0->u[ix][iy][iz];
00081                     u[1] = met0->u[ix + 1][iy][iz];
00082                     u[2] = met0->u[ix][iy + 1][iz];
00083                     u[3] = met0->u[ix + 1][iy + 1][iz];
00084                     u[4] = met0->u[ix][iy][iz + 1];
00085                     u[5] = met0->u[ix + 1][iy][iz + 1];
00086                     u[6] = met0->u[ix][iy + 1][iz + 1];
00087                     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00088
00089                     v[0] = met0->v[ix][iy][iz];
00090                     v[1] = met0->v[ix + 1][iy][iz];
00091                     v[2] = met0->v[ix][iy + 1][iz];
00092                     v[3] = met0->v[ix + 1][iy + 1][iz];
00093                     v[4] = met0->v[ix][iy][iz + 1];
00094                     v[5] = met0->v[ix + 1][iy][iz + 1];
00095                     v[6] = met0->v[ix][iy + 1][iz + 1];
00096                     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00097
00098                     w[0] = (float) (1e3 * DP2DZ(met0->w[ix][iy][iz], met0->p[iz]));
00099                     w[1] = (float) (1e3 * DP2DZ(met0->w[ix + 1][iy][iz], met0->p[iz]));
00100                     w[2] = (float) (1e3 * DP2DZ(met0->w[ix][iy + 1][iz], met0->p[iz]));
00101                     w[3] =
00102                         (float) (1e3 * DP2DZ(met0->w[ix + 1][iy + 1][iz], met0->p[iz]));
00103                     w[4] =
00104                         (float) (1e3 * DP2DZ(met0->w[ix][iy][iz + 1], met0->p[iz + 1]));
00105                     w[5] =
00106                         (float) (1e3 *
00107                             DP2DZ(met0->w[ix + 1][iy][iz + 1], met0->p[iz + 1]));
00108                     w[6] =
00109                         (float) (1e3 *
00110                             DP2DZ(met0->w[ix][iy + 1][iz + 1], met0->p[iz + 1]));
00111                     w[7] =

```

```

00112         (float) (1e3 *
00113                 DP2DZ(met0->w[ix + 1][iy + 1][iz + 1], met0->p[iz + 1]));
00114
00115     /* Collect local wind data... */
00116     u[8] = met1->u[ix][iy][iz];
00117     u[9] = met1->u[ix + 1][iy][iz];
00118     u[10] = met1->u[ix][iy + 1][iz];
00119     u[11] = met1->u[ix + 1][iy + 1][iz];
00120     u[12] = met1->u[ix][iy][iz + 1];
00121     u[13] = met1->u[ix + 1][iy][iz + 1];
00122     u[14] = met1->u[ix][iy + 1][iz + 1];
00123     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00124
00125     v[8] = met1->v[ix][iy][iz];
00126     v[9] = met1->v[ix + 1][iy][iz];
00127     v[10] = met1->v[ix][iy + 1][iz];
00128     v[11] = met1->v[ix + 1][iy + 1][iz];
00129     v[12] = met1->v[ix][iy][iz + 1];
00130     v[13] = met1->v[ix + 1][iy][iz + 1];
00131     v[14] = met1->v[ix][iy + 1][iz + 1];
00132     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00133
00134     w[8] = (float) (1e3 * DP2DZ(met1->w[ix][iy][iz], met1->p[iz]));
00135     w[9] = (float) (1e3 * DP2DZ(met1->w[ix + 1][iy][iz], met1->p[iz]));
00136     w[10] = (float) (1e3 * DP2DZ(met1->w[ix][iy + 1][iz], met1->p[iz]));
00137     w[11] =
00138         (float) (1e3 * DP2DZ(met1->w[ix + 1][iy + 1][iz], met1->p[iz]));
00139     w[12] =
00140         (float) (1e3 * DP2DZ(met1->w[ix][iy][iz + 1], met1->p[iz + 1]));
00141     w[13] =
00142         (float) (1e3 *
00143                 DP2DZ(met1->w[ix + 1][iy][iz + 1], met1->p[iz + 1]));
00144     w[14] =
00145         (float) (1e3 *
00146                 DP2DZ(met1->w[ix][iy + 1][iz + 1], met1->p[iz + 1]));
00147     w[15] =
00148         (float) (1e3 *
00149                 DP2DZ(met1->w[ix + 1][iy + 1][iz + 1], met1->p[iz + 1]));
00150
00151     /* Get standard deviations of local wind data... */
00152     usig[iz][iy] += stddev(u, 16);
00153     vsig[iz][iy] += stddev(v, 16);
00154     wsig[iz][iy] += stddev(w, 16);
00155     n[iz][iy]++;
00156
00157     /* Check surface pressure... */
00158     if (met0->p[iz] > met0->ps[ix][iy]
00159         || met1->p[iz] > met1->ps[ix][iy]) {
00160         usig[iz][iy] = GSL_NAN;
00161         vsig[iz][iy] = GSL_NAN;
00162         wsig[iz][iy] = GSL_NAN;
00163         n[iz][iy] = 0;
00164     }
00165 }
00166 }
00167
00168 /* Create output file... */
00169 LOG(1, "Write subgrid data file: %s", argv[2]);
00170 if (!(out = fopen(argv[2], "w")))
00171     ERRMSG("Cannot create file!");
00172
00173 /* Write header... */
00174 fprintf(out,
00175         "# $1 = time [s]\n"
00176         "# $2 = altitude [km]\n"
00177         "# $3 = longitude [deg]\n"
00178         "# $4 = latitude [deg]\n"
00179         "# $5 = zonal wind standard deviation [m/s]\n"
00180         "# $6 = meridional wind standard deviation [m/s]\n"
00181         "# $7 = vertical velocity standard deviation [m/s]\n"
00182         "# $8 = number of data points\n");
00183
00184 /* Write output... */
00185 for (iy = 0; iy < met0->ny - 1; iy++) {
00186     fprintf(out, "\n");
00187     for (iz = 0; iz < met0->np - 1; iz++)
00188         fprintf(out, "%.2f %g %g %g %g %g %d\n",
00189                 0.5 * (met0->time + met1->time),
00190                 0.5 * (Z(met0->p[iz]) + Z(met1->p[iz + 1])),
00191                 0.0, 0.5 * (met0->lat[iy] + met1->lat[iy + 1]),
00192                 usig[iz][iy] / n[iz][iy], vsig[iz][iy] / n[iz][iy],
00193                 wsig[iz][iy] / n[iz][iy], n[iz][iy]);
00194 }
00195
00196 /* Close file... */
00197 fclose(out);
00198

```

```
00199  /* Free... */
00200  free(clim);
00201  free(met0);
00202  free(met1);
00203
00204  return EXIT_SUCCESS;
00205 }
```

5.39 met_zm.c File Reference

Extract zonal mean from meteorological data.

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

Macros

- `#define NZ 1000`
Maximum number of altitudes.
- `#define NY 721`
Maximum number of latitudes.

Functions

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

5.39.1 Detailed Description

Extract zonal mean from meteorological data.

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

5.39.2 Macro Definition Documentation

5.39.2.1 NZ `#define NZ 1000`

Maximum number of altitudes.

Definition at line [32](#) of file [met_zm.c](#).

5.39.2.2 NY `#define NY 721`

Maximum number of latitudes.

Definition at line [35](#) of file [met_zm.c](#).

5.39.3 Function Documentation

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

Definition at line 41 of file [met_zm.c](#).

```
00043     {
00044
00045     ctl_t ctl;
00046
00047     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00054         usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
00055         pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
00056         capem[NZ][NY], cinm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY],
00057         um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY],
00058         pvm[NZ][NY], o3m[NZ][NY], lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY],
00059         rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
00060         hno3m[NZ][NY], ohm[NZ][NY], h2o2m[NZ][NY], z, z0, z1, dz, zt, tt,
00061         plev[NZ], ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
00062         cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
00063         lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];
00064
00065     static int i, ix, iy, iz, np[NZ][NY], npc[NZ][NY], npt[NZ][NY], ny, nz,
00066         ci[3];
00067
00068     /* Allocate... */
00069     ALLOC(clim, clim_t, 1);
00070     ALLOC(met, met_t, 1);
00071
00072     /* Check arguments... */
00073     if (argc < 4)
00074         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00075
00076     /* Read control parameters... */
00077     read_ctl(argv[1], argc, argv, &ctl);
00078     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00079     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00080     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00081     lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
00082     lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", NULL);
00083     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00084     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00085     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00086
00087     /* Read climatological data... */
00088     read_clim(&ctl, clim);
00089
00090     /* Loop over files... */
00091     for (i = 3; i < argc; i++) {
00092
00093         /* Read meteorological data... */
00094         if (!read_met(argv[i], &ctl, clim, met))
00095             continue;
00096
00097         /* Set vertical grid... */
00098         if (z0 < 0)
00099             z0 = Z(met->p[0]);
00100         if (z1 < 0)
00101             z1 = Z(met->p[met->np - 1]);
00102         nz = 0;
00103         if (dz < 0) {
00104             for (iz = 0; iz < met->np; iz++)
00105                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00106                     plev[nz] = met->p[iz];
00107                     if (++nz > NZ)
00108                         ERRMSG("Too many pressure levels!");
00109                 }
00110             } else
00111                 for (z = z0; z <= z1; z += dz) {
00112                     plev[nz] = P(z);
00113                     if (++nz > NZ)
00114                         ERRMSG("Too many pressure levels!");
00115                 }
```

```

00115     }
00116
00117     /* Set horizontal grid... */
00118     if (dlat <= 0)
00119         dlat = fabs(met->lat[1] - met->lat[0]);
00120     ny = 0;
00121     if (lat0 < -90 && lat1 > 90) {
00122         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00123         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00124     }
00125     for (lat = lat0; lat <= lat1; lat += dlat) {
00126         lats[ny] = lat;
00127         if ((++ny) > NY)
00128             ERRMSG("Too many latitudes!");
00129     }
00130
00131     /* Average... */
00132     for (ix = 0; ix < met->nx; ix++)
00133         if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
00134             for (iy = 0; iy < ny; iy++)
00135                 for (iz = 0; iz < nz; iz++) {
00136
00137                     /* Interpolate meteo data... */
00138                     INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);
00139
00140                     /* Averaging... */
00141                     timem[iz][iy] += met->time;
00142                     lonm[iz][iy] += met->lon[ix];
00143                     zm[iz][iy] += z;
00144                     tm[iz][iy] += t;
00145                     um[iz][iy] += u;
00146                     vm[iz][iy] += v;
00147                     wm[iz][iy] += w;
00148                     pvm[iz][iy] += pv;
00149                     h2om[iz][iy] += h2o;
00150                     o3m[iz][iy] += o3;
00151                     lwcm[iz][iy] += lwc;
00152                     iwc[m][iz][iy] += iwc;
00153                     psm[iz][iy] += ps;
00154                     tsm[iz][iy] += ts;
00155                     zsm[iz][iy] += zs;
00156                     usm[iz][iy] += us;
00157                     vsm[iz][iy] += vs;
00158                     pblm[iz][iy] += pbl;
00159                     pctm[iz][iy] += pct;
00160                     pcbm[iz][iy] += pcb;
00161                     clm[iz][iy] += cl;
00162                     if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00163                         && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00164                         plclm[iz][iy] += plcl;
00165                         plfcm[iz][iy] += plfc;
00166                         pelm[iz][iy] += pel;
00167                         capem[iz][iy] += cape;
00168                         cinm[iz][iy] += cin;
00169                         npc[iz][iy]++;
00170                     }
00171                     if (gsl_finite(pt)) {
00172                         ptm[iz][iy] += pt;
00173                         ztm[iz][iy] += zt;
00174                         ttm[iz][iy] += tt;
00175                         h2otm[iz][iy] += h2ot;
00176                         npt[iz][iy]++;
00177                     }
00178                     rhm[iz][iy] += RH(plev[iz], t, h2o);
00179                     rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
00180                     tdewm[iz][iy] += TDEW(plev[iz], h2o);
00181                     ticem[iz][iy] += TICE(plev[iz], h2o);
00182                     hno3m[iz][iy] += clim_hno3(clim, met->time, lats[iy], plev[iz]);
00183                     tnatm[iz][iy] +=
00184                         nat_temperature(plev[iz], h2o,
00185                                         clim_hno3(clim, met->time, lats[iy], plev[iz]));
00186                     ohm[iz][iy] +=
00187                         clim_oh_diurnal(&ctl, clim, met->time, plev[iz], met->lon[ix],
00188                                         lats[iy]);
00189                     h2o2m[iz][iy] += clim_h2o2(clim, met->time, lats[iy], plev[iz]);
00190                     np[iz][iy]++;
00191                 }
00192     }
00193
00194     /* Create output file... */
00195     LOG(1, "Write meteorological data file: %s", argv[2]);
00196     if (!(out = fopen(argv[2], "w")))
00197         ERRMSG("Cannot create file!");
00198
00199     /* Write header... */
00200     fprintf(out,
00201         "# $1 = time [s]\n"

```



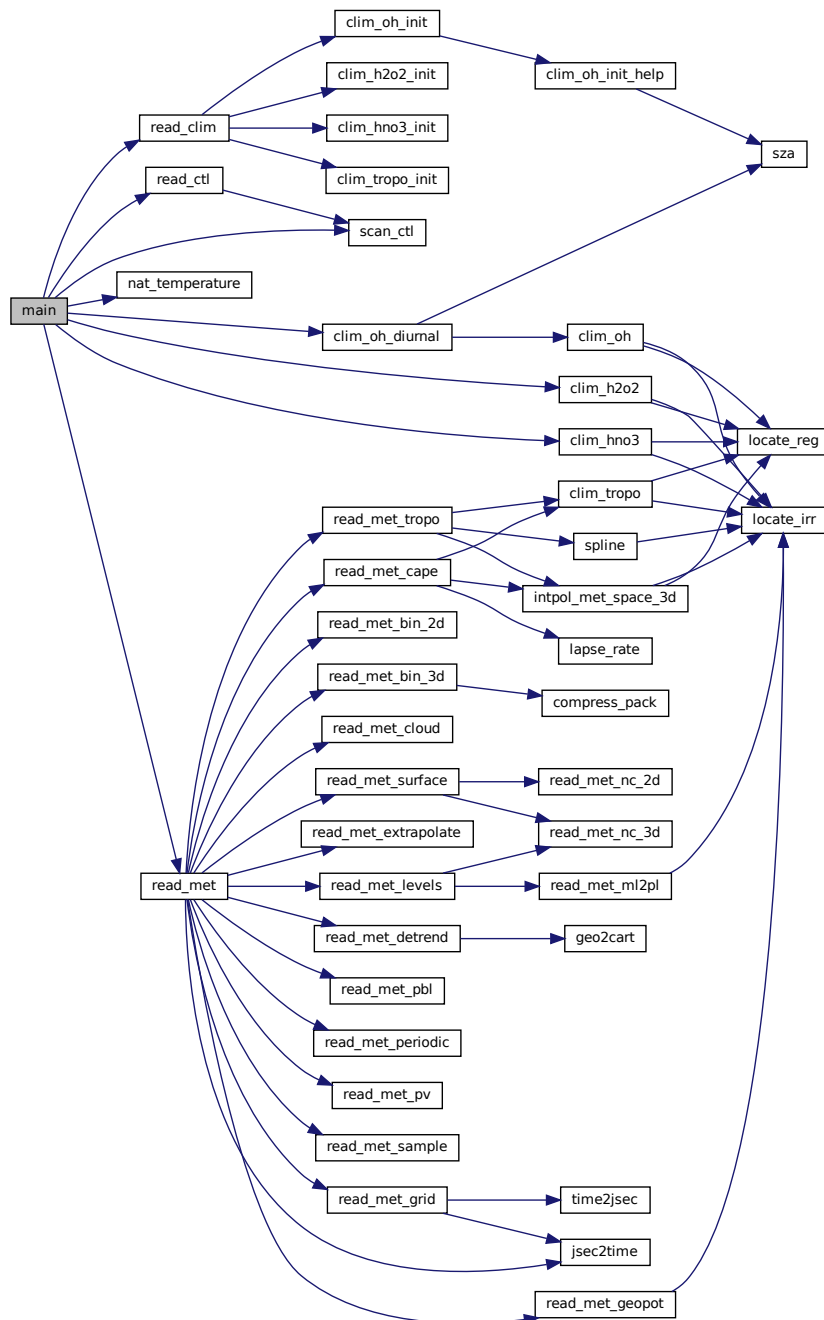
```

00202     "# $2 = altitude [km]\n"
00203     "# $3 = longitude [deg]\n"
00204     "# $4 = latitude [deg]\n"
00205     "# $5 = pressure [hPa]\n"
00206     "# $6 = temperature [K]\n"
00207     "# $7 = zonal wind [m/s]\n"
00208     "# $8 = meridional wind [m/s]\n"
00209     "# $9 = vertical velocity [hPa/s]\n"
00210     "# $10 = H2O volume mixing ratio [ppv]\n");
00211 fprintf(out,
00212     "# $11 = O3 volume mixing ratio [ppv]\n"
00213     "# $12 = geopotential height [km]\n"
00214     "# $13 = potential vorticity [PVU]\n"
00215     "# $14 = surface pressure [hPa]\n"
00216     "# $15 = surface temperature [K]\n"
00217     "# $16 = surface geopotential height [km]\n"
00218     "# $17 = surface zonal wind [m/s]\n"
00219     "# $18 = surface meridional wind [m/s]\n"
00220     "# $19 = tropopause pressure [hPa]\n"
00221     "# $20 = tropopause geopotential height [km]\n");
00222 fprintf(out,
00223     "# $21 = tropopause temperature [K]\n"
00224     "# $22 = tropopause water vapor [ppv]\n"
00225     "# $23 = cloud liquid water content [kg/kg]\n"
00226     "# $24 = cloud ice water content [kg/kg]\n"
00227     "# $25 = total column cloud water [kg/m^2]\n"
00228     "# $26 = cloud top pressure [hPa]\n"
00229     "# $27 = cloud bottom pressure [hPa]\n"
00230     "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00231     "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00232     "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00233 fprintf(out,
00234     "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00235     "# $32 = convective inhibition (CIN) [J/kg]\n"
00236     "# $33 = relative humidity over water [%]\n"
00237     "# $34 = relative humidity over ice [%]\n"
00238     "# $35 = dew point temperature [K]\n"
00239     "# $36 = frost point temperature [K]\n"
00240     "# $37 = NAT temperature [K]\n"
00241     "# $38 = HNO3 volume mixing ratio [ppv]\n"
00242     "# $39 = OH concentration [molec/cm^3]\n"
00243     "# $40 = H2O2 concentration [molec/cm^3]\n");
00244 fprintf(out,
00245     "# $41 = boundary layer pressure [hPa]\n"
00246     "# $42 = number of data points\n"
00247     "# $43 = number of tropopause data points\n"
00248     "# $44 = number of CAPE data points\n");
00249
00250 /* Write data... */
00251 for (iz = 0; iz < nz; iz++) {
00252     fprintf(out, "\n");
00253     for (iy = 0; iy < ny; iy++)
00254         fprintf(out,
00255             "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00256             " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00257             " %g %g %g %g %g %d %d %d\n",
00258             timem[iz][iy] / np[iz][iy], Z(plev[iz]),
00259             lonm[iz][iy] / np[iz][iy], lats[iy],
00260             plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00261             vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00262             h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00263             zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00264             psm[iz][iy] / np[iz][iy], tsm[iz][iy] / np[iz][iy],
00265             zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
00266             vsm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00267             ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
00268             h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00269             icwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
00270             pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],
00271             plclm[iz][iy] / npc[iz][iy], plfcm[iz][iy] / npc[iz][iy],
00272             pelm[iz][iy] / npc[iz][iy], capem[iz][iy] / npc[iz][iy],
00273             cinm[iz][iy] / npc[iz][iy], rhm[iz][iy] / np[iz][iy],
00274             rhicem[iz][iy] / np[iz][iy], tdewm[iz][iy] / np[iz][iy],
00275             ticem[iz][iy] / np[iz][iy], tnatm[iz][iy] / np[iz][iy],
00276             hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
00277             h2o2m[iz][iy] / np[iz][iy], pblm[iz][iy] / np[iz][iy],
00278             np[iz][iy], npt[iz][iy], npc[iz][iy]);
00279 }
00280
00281 /* Close file... */
00282 fclose(out);
00283
00284 /* Free... */
00285 free(clim);
00286 free(met);
00287
00288 return EXIT_SUCCESS;

```

00289 }

Here is the call graph for this function:



5.40 met_zm.c

```

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

```

```

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) 2013-2022 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     clim_t *clim;
00048
00049     met_t *met;
00050
00051     FILE *out;
00052
00053     static double timem[NZ][NY], psm[NZ][NY], tsm[NZ][NY], zsm[NZ][NY],
00054         usm[NZ][NY], vsm[NZ][NY], pblm[NZ][NY], ptm[NZ][NY], pctm[NZ][NY],
00055         pcbm[NZ][NY], clm[NZ][NY], plclm[NZ][NY], plfcm[NZ][NY], pelm[NZ][NY],
00056         capem[NZ][NY], cinm[NZ][NY], ttcm[NZ][NY], ztm[NZ][NY], tm[NZ][NY],
00057         um[NZ][NY], vm[NZ][NY], wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY],
00058         pvm[NZ][NY], o3m[NZ][NY], lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY],
00059         rhm[NZ][NY], rhicem[NZ][NY], tdewm[NZ][NY], ticem[NZ][NY], tnatm[NZ][NY],
00060         hno3m[NZ][NY], ohm[NZ][NY], h2o2m[NZ][NY], z, z0, z1, dz, zt, tt,
00061         plev[NZ], ps, ts, zs, us, vs, pbl, pt, pct, pcb, plcl, plfc, pel,
00062         cape, cin, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc,
00063         lat, lat0, lat1, dlat, lats[NY], lon0, lon1, lonm[NZ][NY], cw[3];
00064
00065     static int i, ix, iy, iz, np[NZ][NY], npc[NZ][NY], npt[NZ][NY], ny, nz,
00066         ci[3];
00067
00068     /* Allocate... */
00069     ALLOC(clim, clim_t, 1);
00070     ALLOC(met, met_t, 1);
00071
00072     /* Check arguments... */
00073     if (argc < 4)
00074         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00075
00076     /* Read control parameters... */
00077     read_ctl(argv[1], argc, argv, &ctl);
00078     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00079     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00080     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00081     lon0 = scan_ctl(argv[1], argc, argv, "ZM_LON0", -1, "-360", NULL);
00082     lon1 = scan_ctl(argv[1], argc, argv, "ZM_LON1", -1, "360", NULL);
00083     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00084     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00085     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00086
00087     /* Read climatological data... */
00088     read_clim(&ctl, clim);
00089
00090     /* Loop over files... */
00091     for (i = 3; i < argc; i++) {
00092
00093         /* Read meteorological data... */
00094         if (!read_met(argv[i], &ctl, clim, met))
00095             continue;
00096
00097         /* Set vertical grid... */
00098         if (z0 < 0)
00099             z0 = Z(met->p[0]);
00100         if (z1 < 0)
00101             z1 = Z(met->p[met->np - 1]);

```

```

00102     nz = 0;
00103     if (dz < 0) {
00104         for (iz = 0; iz < met->np; iz++)
00105             if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00106                 plev[nz] = met->p[iz];
00107                 if ((++nz) > NZ)
00108                     ERRMSG("Too many pressure levels!");
00109             }
00110     } else
00111         for (z = z0; z <= z1; z += dz) {
00112             plev[nz] = P(z);
00113             if ((++nz) > NZ)
00114                 ERRMSG("Too many pressure levels!");
00115         }
00116
00117     /* Set horizontal grid... */
00118     if (dlat <= 0)
00119         dlat = fabs(met->lat[1] - met->lat[0]);
00120     ny = 0;
00121     if (lat0 < -90 && lat1 > 90) {
00122         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00123         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00124     }
00125     for (lat = lat0; lat <= lat1; lat += dlat) {
00126         lats[ny] = lat;
00127         if ((++ny) > NY)
00128             ERRMSG("Too many latitudes!");
00129     }
00130
00131     /* Average... */
00132     for (ix = 0; ix < met->nx; ix++)
00133         if (met->lon[ix] >= lon0 && met->lon[ix] <= lon1)
00134             for (iy = 0; iy < ny; iy++)
00135                 for (iz = 0; iz < nz; iz++) {
00136
00137                     /* Interpolate meteo data... */
00138                     INTPOL_SPACE_ALL(plev[iz], met->lon[ix], lats[iy]);
00139
00140                     /* Averaging... */
00141                     timem[iz][iy] += met->time;
00142                     lonm[iz][iy] += met->lon[ix];
00143                     zm[iz][iy] += z;
00144                     tm[iz][iy] += t;
00145                     um[iz][iy] += u;
00146                     vm[iz][iy] += v;
00147                     wm[iz][iy] += w;
00148                     pvm[iz][iy] += pv;
00149                     h2om[iz][iy] += h2o;
00150                     o3m[iz][iy] += o3;
00151                     lwcm[iz][iy] += lwc;
00152                     iwcm[iz][iy] += iwc;
00153                     psm[iz][iy] += ps;
00154                     tsm[iz][iy] += ts;
00155                     zsm[iz][iy] += zs;
00156                     usm[iz][iy] += us;
00157                     vsm[iz][iy] += vs;
00158                     pblm[iz][iy] += pbl;
00159                     pctm[iz][iy] += pct;
00160                     pcbm[iz][iy] += pcb;
00161                     clm[iz][iy] += cl;
00162                     if (gsl_finite(plfc) && gsl_finite(pel) && cape >= ctl.conv_cape
00163                         && (ctl.conv_cin <= 0 || cin < ctl.conv_cin)) {
00164                         plclm[iz][iy] += plcl;
00165                         plfcm[iz][iy] += plfc;
00166                         pelm[iz][iy] += pel;
00167                         capem[iz][iy] += cape;
00168                         cinm[iz][iy] += cin;
00169                         npc[iz][iy]++;
00170                     }
00171                     if (gsl_finite(pt)) {
00172                         ptm[iz][iy] += pt;
00173                         ztm[iz][iy] += zt;
00174                         ttm[iz][iy] += tt;
00175                         h2otm[iz][iy] += h2ot;
00176                         npt[iz][iy]++;
00177                     }
00178                     rhm[iz][iy] += RH(plev[iz], t, h2o);
00179                     rhicem[iz][iy] += RHICE(plev[iz], t, h2o);
00180                     tdewm[iz][iy] += TDEW(plev[iz], h2o);
00181                     ticem[iz][iy] += TICE(plev[iz], h2o);
00182                     hno3m[iz][iy] += clim_hno3(clim, met->time, lats[iy], plev[iz]);
00183                     tnatm[iz][iy] +=
00184                         nat_temperature(plev[iz], h2o,
00185                                         clim_hno3(clim, met->time, lats[iy], plev[iz]));
00186                     ohm[iz][iy] +=
00187                         clim_oh_diurnal(&ctl, clim, met->time, plev[iz], met->lon[ix],
00188                                         lats[iy]);

```

```
00189 h2o2m[iz][iy] += clim_h2o2(clim, met->time, lats[iy], plev[iy]);
00190 np[iz][iy]++;
00191 }
00192 }
00193
00194 /* Create output file... */
00195 LOG(1, "Write meteorological data file: %s", argv[2]);
00196 if (!out = fopen(argv[2], "w"))
00197     ERRMSG("Cannot create file!");
00198
00199 /* Write header... */
00200 fprintf(out,
00201         "# $1 = time [s]\n"
00202         "# $2 = altitude [km]\n"
00203         "# $3 = longitude [deg]\n"
00204         "# $4 = latitude [deg]\n"
00205         "# $5 = pressure [hPa]\n"
00206         "# $6 = temperature [K]\n"
00207         "# $7 = zonal wind [m/s]\n"
00208         "# $8 = meridional wind [m/s]\n"
00209         "# $9 = vertical velocity [hPa/s]\n"
00210         "# $10 = H2O volume mixing ratio [ppv]\n");
00211 fprintf(out,
00212         "# $11 = O3 volume mixing ratio [ppv]\n"
00213         "# $12 = geopotential height [km]\n"
00214         "# $13 = potential vorticity [PVU]\n"
00215         "# $14 = surface pressure [hPa]\n"
00216         "# $15 = surface temperature [K]\n"
00217         "# $16 = surface geopotential height [km]\n"
00218         "# $17 = surface zonal wind [m/s]\n"
00219         "# $18 = surface meridional wind [m/s]\n"
00220         "# $19 = tropopause pressure [hPa]\n"
00221         "# $20 = tropopause geopotential height [km]\n");
00222 fprintf(out,
00223         "# $21 = tropopause temperature [K]\n"
00224         "# $22 = tropopause water vapor [ppv]\n"
00225         "# $23 = cloud liquid water content [kg/kg]\n"
00226         "# $24 = cloud ice water content [kg/kg]\n"
00227         "# $25 = total column cloud water [kg/m^2]\n"
00228         "# $26 = cloud top pressure [hPa]\n"
00229         "# $27 = cloud bottom pressure [hPa]\n"
00230         "# $28 = pressure at lifted condensation level (LCL) [hPa]\n"
00231         "# $29 = pressure at level of free convection (LFC) [hPa]\n"
00232         "# $30 = pressure at equilibrium level (EL) [hPa]\n");
00233 fprintf(out,
00234         "# $31 = convective available potential energy (CAPE) [J/kg]\n"
00235         "# $32 = convective inhibition (CIN) [J/kg]\n"
00236         "# $33 = relative humidity over water [%]\n"
00237         "# $34 = relative humidity over ice [%]\n"
00238         "# $35 = dew point temperature [K]\n"
00239         "# $36 = frost point temperature [K]\n"
00240         "# $37 = NAT temperature [K]\n"
00241         "# $38 = HNO3 volume mixing ratio [ppv]\n"
00242         "# $39 = OH concentration [molec/cm^3]\n"
00243         "# $40 = H2O2 concentration [molec/cm^3]\n");
00244 fprintf(out,
00245         "# $41 = boundary layer pressure [hPa]\n"
00246         "# $42 = number of data points\n"
00247         "# $43 = number of tropopause data points\n"
00248         "# $44 = number of CAPE data points\n");
00249
00250 /* Write data... */
00251 for (iz = 0; iz < nz; iz++) {
00252     fprintf(out, "\n");
00253     for (iy = 0; iy < ny; iy++)
00254         fprintf(out,
00255                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00256                 " %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n"
00257                 " %g %g %g %g %g %d %d %d\n",
00258                 timem[iz][iy] / np[iz][iy], Z(plev[iz]),
00259                 lonm[iz][iy] / np[iz][iy], lats[iy],
00260                 plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00261                 vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00262                 h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00263                 zm[iz][iy] / np[iz][iy], pvmm[iz][iy] / np[iz][iy],
00264                 psm[iz][iy] / np[iz][iy], tsm[iz][iy] / np[iz][iy],
00265                 zsm[iz][iy] / np[iz][iy], usm[iz][iy] / np[iz][iy],
00266                 vsm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00267                 ztm[iz][iy] / npt[iz][iy], ttmm[iz][iy] / npt[iz][iy],
00268                 h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00269                 iwcm[iz][iy] / np[iz][iy], clmm[iz][iy] / np[iz][iy],
00270                 pctm[iz][iy] / np[iz][iy], pcbm[iz][iy] / np[iz][iy],
00271                 plclm[iz][iy] / npc[iz][iy], plfcmm[iz][iy] / npc[iz][iy],
00272                 pelmm[iz][iy] / npc[iz][iy], capemm[iz][iy] / npc[iz][iy],
00273                 cinm[iz][iy] / npc[iz][iy], rhmm[iz][iy] / np[iz][iy],
00274                 rhicem[iz][iy] / np[iz][iy], tdewmm[iz][iy] / np[iz][iy],
00275                 ticem[iz][iy] / np[iz][iy], tnatmm[iz][iy] / np[iz][iy],
```

```

00276             hno3m[iz][iy] / np[iz][iy], ohm[iz][iy] / np[iz][iy],
00277             h2o2m[iz][iy] / np[iz][iy], pblm[iz][iy] / np[iz][iy],
00278             np[iz][iy], npt[iz][iy], npc[iz][iy]);
00279     }
00280
00281     /* Close file... */
00282     fclose(out);
00283
00284     /* Free... */
00285     free(clim);
00286     free(met);
00287
00288     return EXIT_SUCCESS;
00289 }

```

5.41 sedi.c File Reference

Calculate sedimentation velocity.

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

Functions

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

5.41.1 Detailed Description

Calculate sedimentation velocity.

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

5.41.2 Function Documentation

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

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

```

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     /* Write output... */
00056     printf("    p= %g hPa\n", p);
00057     printf("    T= %g K\n", T);
00058     printf("    r_p= %g microns\n", r_p);
00059     printf("rho_p= %g kg/m^3\n", rho_p);
00060     printf("rho_a= %g kg/m^3\n", RHO(p, T));
00061     printf("    v_s= %g m/s\n", vs);
00062     printf("    Re= %g\n", Re);
00063
00064     return EXIT_SUCCESS;
00065 }

```

Here is the call graph for this function:



5.42 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 /* Write output... */
00056 printf("    p= %g hPa\n", p);
00057 printf("    T= %g K\n", T);
00058 printf("    r_p= %g microns\n", r_p);
00059 printf("rho_p= %g kg/m^3\n", rho_p);
00060 printf("rho_a= %g kg/m^3\n", RHO(p, T));
00061 printf("    v_s= %g m/s\n", vs);
00062 printf("    Re= %g\n", Re);
00063
00064 return EXIT_SUCCESS;
00065 }
```

5.43 time2jsec.c File Reference

Convert date to Julian seconds.

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

Functions

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

5.43.1 Detailed Description

Convert date to Julian seconds.

Definition in file [time2jsec.c](#).

5.43.2 Function Documentation

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

Definition at line 27 of file [time2jsec.c](#).

```
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:



5.44 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 }

```

5.45 tnat.c File Reference

Calculate PSC temperatures.

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

Functions

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

5.45.1 Detailed Description

Calculate PSC temperatures.

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

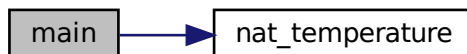
5.45.2 Function Documentation

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

Definition at line 31 of file [tnat.c](#).

```
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     /* Write output... */
00045     printf("      p= %g hPa\n", p);
00046     printf("  q_H2O= %g ppv\n", h2o);
00047     printf("q_HNO3= %g ppv\n", hno3);
00048     printf("  T_dew= %g K\n", TDEW(p, h2o));
00049     printf("  T_ice= %g K\n", TICE(p, h2o));
00050     printf("  T_NAT= %g K\n", nat_temperature(p, h2o, hno3));
00051
00052     return EXIT_SUCCESS;
00053 }
```

Here is the call graph for this function:



5.46 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) 2013-2023 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     /* Write output... */
00045     printf("      p= %g hPa\n", p);
00046     printf("  q_H2O= %g ppv\n", h2o);
00047     printf("  q_HNO3= %g ppv\n", hno3);
00048     printf("  T_dew= %g K\n", TDEW(p, h2o));
00049     printf("  T_ice= %g K\n", TICE(p, h2o));
00050     printf("  T_NAT= %g K\n", nat_temperature(p, h2o, hno3));
00051
00052     return EXIT_SUCCESS;
00053 }

```

5.47 trac.c File Reference

Lagrangian particle dispersion model.

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

Functions

- void `module_advect` (`ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt)
Calculate advection of air parcels.
- void `module_bound_cond` (`ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt)
Apply boundary conditions.
- void `module_convection` (`ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt, double *rs)
Calculate convection of air parcels.
- void `module_decay` (`ctl_t` *ctl, `clim_t` *clim, `atm_t` *atm, double *dt)
Calculate exponential decay of particle mass.
- void `module_diffusion_meso` (`ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, `cache_t` *cache, double *dt, double *rs)

- *Calculate mesoscale diffusion.*
- void [module_diffusion_turb](#) ([ctl_t](#) *ctl, [clim_t](#) *clim, [atm_t](#) *atm, double *dt, double *rs)
- *Calculate turbulent diffusion.*
- void [module_dry_deposition](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- *Calculate dry deposition.*
- void [module_isosurf_init](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache)
- *Initialize isosurface module.*
- void [module_isosurf](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache)
- *Force air parcels to stay on isosurface.*
- void [module_meteo](#) ([ctl_t](#) *ctl, [clim_t](#) *clim, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm)
- *Interpolate meteo data for air parcel positions.*
- void [module_oh_chem](#) ([ctl_t](#) *ctl, [clim_t](#) *clim, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- *Calculate OH chemistry.*
- void [module_h2o2_chem](#) ([ctl_t](#) *ctl, [clim_t](#) *clim, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt, double *rs)
- *Calculate H2O2 chemistry.*
- void [module_chemgrid](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
- *Interpolate to chemistry grid.*
- void [module_position](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- *Check position of air parcels.*
- void [module_rng_init](#) (int ntask)
- *Initialize random number generator...*
- void [module_rng](#) (double *rs, size_t n, int method)
- *Generate random numbers.*
- void [module_sedi](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- *Calculate sedimentation of air parcels.*
- void [module_sort](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [atm_t](#) *atm)
- *Sort particles according to box index.*
- void [module_sort_help](#) (double *a, int *p, int np)
- *Helper function for sorting module.*
- void [module_timesteps](#) ([ctl_t](#) *ctl, [atm_t](#) *atm, [met_t](#) *met0, double *dt, double t)
- *Calculate time steps.*
- void [module_timesteps_init](#) ([ctl_t](#) *ctl, [atm_t](#) *atm)
- *Initialize timesteps.*
- void [module_wet_deposition](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- *Calculate wet deposition.*
- void [write_output](#) (const char *dirname, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
- *Write simulation output.*
- int [main](#) (int argc, char *argv[])

5.47.1 Detailed Description

Lagrangian particle dispersion model.

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

5.47.2 Function Documentation

5.47.2.1 module_advect() void module_advect (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate advection of air parcels.

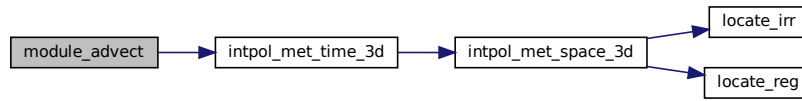
Definition at line 558 of file [trac.c](#).

```

00563     {
00564
00565     /* Set timer... */
00566     SELECT_TIMER("MODULE_ADEVECTION", "PHYSICS", NVTX_GPU);
00567
00568     const int np = atm->np;
00569 #ifdef _OPENACC
00570 #pragma acc data present(ctl,met0,met1,atm,dt)
00571 #pragma acc parallel loop independent gang vector
00572 #else
00573 #pragma omp parallel for default(shared)
00574 #endif
00575     for (int ip = 0; ip < np; ip++)
00576         if (dt[ip] != 0) {
00577
00578         /* Init... */
00579         double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];
00580
00581         /* Loop over integration nodes... */
00582         for (int i = 0; i < ctl->advect; i++) {
00583
00584         /* Set position... */
00585         if (i == 0) {
00586             dts = 0.0;
00587             x[0] = atm->lon[ip];
00588             x[1] = atm->lat[ip];
00589             x[2] = atm->p[ip];
00590         } else {
00591             dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
00592             x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
00593             x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
00594             x[2] = atm->p[ip] + dts * w[i - 1];
00595         }
00596         double tm = atm->time[ip] + dts;
00597
00598         /* Interpolate meteo data... */
00599 #ifdef UVW
00600         intpol_met_time_uvw(met0, met1, tm, x[2], x[0], x[1],
00601                             &u[i], &v[i], &w[i]);
00602 #else
00603         INTPOL_INIT;
00604         intpol_met_time_3d(met0, met0->u, met1, met1->u, tm,
00605                           x[2], x[0], x[1], &u[i], ci, cw, 1);
00606         intpol_met_time_3d(met0, met0->v, met1, met1->v, tm,
00607                           x[2], x[0], x[1], &v[i], ci, cw, 0);
00608         intpol_met_time_3d(met0, met0->w, met1, met1->w, tm,
00609                           x[2], x[0], x[1], &w[i], ci, cw, 0);
00610 #endif
00611
00612         /* Get mean wind... */
00613         double k = 1.0;
00614         if (ctl->advect == 2)
00615             k = (i == 0 ? 0.0 : 1.0);
00616         else if (ctl->advect == 4)
00617             k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
00618         um += k * u[i];
00619         vm += k * v[i];
00620         wm += k * w[i];
00621     }
00622
00623     /* Set new position... */
00624     atm->time[ip] += dt[ip];
00625     atm->lon[ip] += DX2DEG(dt[ip] * um / 1000.,
00626                           (ctl->advect == 2 ? x[1] : atm->lat[ip]));
00627     atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
00628     atm->p[ip] += dt[ip] * wm;
00629 }
00630 }

```

Here is the call graph for this function:



5.47.2.2 module_bound_cond() void module_bound_cond (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Apply boundary conditions.

Definition at line 634 of file [trac.c](#).

```

00639     {
00640
00641     /* Set timer... */
00642     SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);
00643
00644     /* Check quantity flags... */
00645     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00646         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00647
00648     const int np = atm->np;
00649 #ifdef _OPENACC
00650 #pragma acc data present(ctl, met0, met1, atm, dt)
00651 #pragma acc parallel loop independent gang vector
00652 #else
00653 #pragma omp parallel for default(shared)
00654 #endif
00655     for (int ip = 0; ip < np; ip++)
00656         if (dt[ip] != 0) {
00657
00658             /* Check latitude and pressure range... */
00659             if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
00660                 || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
00661                 continue;
00662
00663             /* Check surface layer... */
00664             if (ctl->bound_dps > 0 || ctl->bound_dzs > 0
00665                 || ctl->bound_zetas > 0 || ctl->bound_pbl) {
00666
00667                 /* Get surface pressure... */
00668                 double ps;
00669                 INTPOL_INIT;
00670                 INTPOL_2D(ps, 1);
00671
00672                 /* Check pressure... */
00673                 if (ctl->bound_dps > 0 && atm->p[ip] < ps - ctl->bound_dps)
00674                     continue;
00675
00676                 /* Check height... */
00677                 if (ctl->bound_dzs > 0 && Z(atm->p[ip]) > Z(ps) + ctl->bound_dzs)
00678                     continue;
00679
00680                 /* Check zeta range... */
00681                 if (ctl->bound_zetas > 0) {
00682                     double t;
00683                     INTPOL_3D(t, 1);
00684                     if (ZETA(ps, atm->p[ip], t) > ctl->bound_zetas)
00685                         continue;
00686                 }
00687
00688                 /* Check planetary boundary layer... */

```

```

00689         if (ctl->bound_pbl) {
00690             double pbl;
00691             INTPOL_2D(pbl, 0);
00692             if (atm->p[ip] < pbl)
00693                 continue;
00694         }
00695     }
00696
00697     /* Set mass and volume mixing ratio... */
00698     if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
00699         atm->q[ctl->qnt_m][ip] =
00700             ctl->bound_mass + ctl->bound_mass_trend * atm->time[ip];
00701     if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
00702         atm->q[ctl->qnt_vmr][ip] =
00703             ctl->bound_vmr + ctl->bound_vmr_trend * atm->time[ip];
00704 }
00705 }

```

5.47.2.3 module_convection() void module_convection (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate convection of air parcels.

Definition at line 709 of file [trac.c](#).

```

00715     {
00716
00717         /* Set timer... */
00718         SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVIX_GPU);
00719
00720         /* Create random numbers... */
00721         module_rng(rs, (size_t) atm->np, 0);
00722
00723         const int np = atm->np;
00724         #ifndef _OPENACC
00725         #pragma acc data present(ctl, met0, met1, atm, dt, rs)
00726         #pragma acc parallel loop independent gang vector
00727         #else
00728         #pragma omp parallel for default(shared)
00729         #endif
00730         for (int ip = 0; ip < np; ip++)
00731             if (dt[ip] != 0) {
00732
00733                 double cape, cin, pel, ps;
00734
00735                 /* Interpolate CAPE... */
00736                 INTPOL_INIT;
00737                 INTPOL_2D(cape, 1);
00738
00739                 /* Check threshold... */
00740                 if (isfinite(cape) && cape >= ctl->conv_cape) {
00741
00742                     /* Check CIN... */
00743                     if (ctl->conv_cin > 0) {
00744                         INTPOL_2D(cin, 0);
00745                         if (isfinite(cin) && cin >= ctl->conv_cin)
00746                             continue;
00747                     }
00748
00749                     /* Interpolate equilibrium level... */
00750                     INTPOL_2D(pel, 0);
00751
00752                     /* Check whether particle is above cloud top... */
00753                     if (!isfinite(pel) || atm->p[ip] < pel)
00754                         continue;
00755
00756                     /* Set pressure range for mixing... */
00757                     double pbot = atm->p[ip];
00758                     double ptop = atm->p[ip];
00759                     if (ctl->conv_mix_bot == 1) {
00760                         INTPOL_2D(ps, 0);
00761                         pbot = ps;

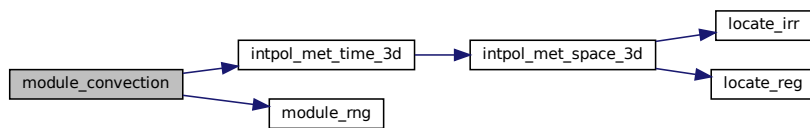
```

```

00762     }
00763     if (ctl->conv_mix_top == 1)
00764         ptop = pel;
00765
00766     /* Vertical mixing based on pressure... */
00767     if (ctl->conv_mix == 0)
00768         atm->p[ip] = pbot + (ptop - pbot) * rs[ip];
00769
00770     /* Vertical mixing based on density... */
00771     else if (ctl->conv_mix == 1) {
00772
00773         /* Get density range... */
00774         double tbot, ttop;
00775         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00776             pbot, atm->lon[ip], atm->lat[ip], &tbot,
00777             ci, cw, 1);
00778         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00779             ptop, atm->lon[ip], atm->lat[ip], &ttop,
00780             ci, cw, 1);
00781         double rhobot = pbot / tbot;
00782         double rhotop = ptop / ttop;
00783
00784         /* Get new density... */
00785         double lrho = log(rhobot + (rhotop - rhobot) * rs[ip]);
00786
00787         /* Find pressure... */
00788         double lrhobot = log(rhobot);
00789         double lrtop = log(rhotop);
00790         double lpbob = log(pbot);
00791         double lptop = log(ptop);
00792         atm->p[ip] = exp(LIN(lrhobot, lpbob, lrtop, lptop, lrho));
00793     }
00794 }
00795 }
00796 }

```

Here is the call graph for this function:



5.47.2.4 module_decay() void module_decay (

```

    ctl_t * ctl,
    clim_t * clim,
    atm_t * atm,
    double * dt )

```

Calculate exponential decay of particle mass.

Definition at line 800 of file [trac.c](#).

```

00804     {
00805
00806         /* Set timer... */
00807         SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);
00808
00809         /* Check quantity flags... */
00810         if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00811             ERRMSG("Module needs quantity mass or volume mixing ratio!");
00812
00813         const int np = atm->np;
00814         #ifdef _OPENACC
00815         #pragma acc data present(ctl,clim,atm,dt)
00816         #pragma acc parallel loop independent gang vector
00817         #else

```

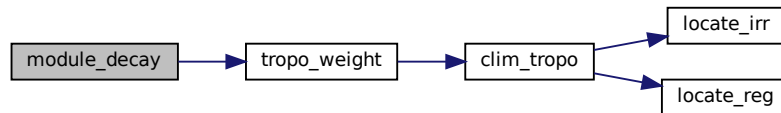


```

00818 #pragma omp parallel for default(shared)
00819 #endif
00820 for (int ip = 0; ip < np; ip++)
00821     if (dt[ip] != 0) {
00822
00823         /* Get weighting factor... */
00824         double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00825
00826         /* Set lifetime... */
00827         double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00828
00829         /* Calculate exponential decay... */
00830         double aux = exp(-dt[ip] / tdec);
00831         if (ctl->qnt_m >= 0) {
00832             if (ctl->qnt_mloss_decay >= 0)
00833                 atm->q[ctl->qnt_mloss_decay][ip]
00834                 += atm->q[ctl->qnt_m][ip] * (1 - aux);
00835             atm->q[ctl->qnt_m][ip] *= aux;
00836         }
00837         if (ctl->qnt_vmr >= 0)
00838             atm->q[ctl->qnt_vmr][ip] *= aux;
00839     }
00840 }

```

Here is the call graph for this function:



5.47.2.5 module_diffusion_meso() void module_diffusion_meso (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    cache_t * cache,
    double * dt,
    double * rs )

```

Calculate mesoscale diffusion.

Definition at line 844 of file [trac.c](#).

```

00851     {
00852
00853         /* Set timer... */
00854         SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);
00855
00856         /* Create random numbers... */
00857         module_rng(rs, 3 * (size_t) atm->np, 1);
00858
00859         const int np = atm->np;
00860 #ifdef _OPENACC
00861 #pragma acc data present(ctl, met0, met1, atm, cache, dt, rs)
00862 #pragma acc parallel loop independent gang vector
00863 #else
00864 #pragma omp parallel for default(shared)
00865 #endif
00866         for (int ip = 0; ip < np; ip++)
00867             if (dt[ip] != 0) {
00868
00869                 /* Get indices... */
00870                 int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);

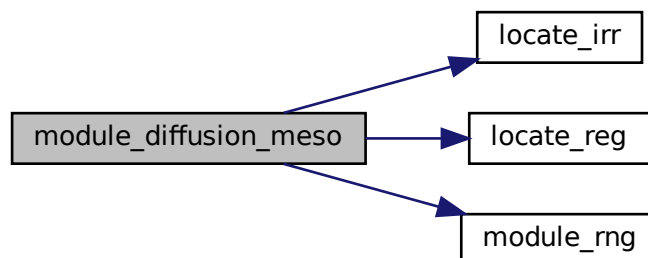
```

```

00871     int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00872     int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00873
00874     /* Get standard deviations of local wind data... */
00875     float umean = 0, usig = 0, vmean = 0, vsig = 0, wmean = 0, wsig = 0;
00876     for (int i = 0; i < 2; i++)
00877         for (int j = 0; j < 2; j++)
00878             for (int k = 0; k < 2; k++) {
00879 #ifdef UVW
00880         umean += met0->uvw[ix + i][iy + j][iz + k][0];
00881         usig += SQR(met0->uvw[ix + i][iy + j][iz + k][0]);
00882         vmean += met0->uvw[ix + i][iy + j][iz + k][1];
00883         vsig += SQR(met0->uvw[ix + i][iy + j][iz + k][1]);
00884         wmean += met0->uvw[ix + i][iy + j][iz + k][2];
00885         wsig += SQR(met0->uvw[ix + i][iy + j][iz + k][2]);
00886
00887         umean += met1->uvw[ix + i][iy + j][iz + k][0];
00888         usig += SQR(met1->uvw[ix + i][iy + j][iz + k][0]);
00889         vmean += met1->uvw[ix + i][iy + j][iz + k][1];
00890         vsig += SQR(met1->uvw[ix + i][iy + j][iz + k][1]);
00891         wmean += met1->uvw[ix + i][iy + j][iz + k][2];
00892         wsig += SQR(met1->uvw[ix + i][iy + j][iz + k][2]);
00893 #else
00894         umean += met0->u[ix + i][iy + j][iz + k];
00895         usig += SQR(met0->u[ix + i][iy + j][iz + k]);
00896         vmean += met0->v[ix + i][iy + j][iz + k];
00897         vsig += SQR(met0->v[ix + i][iy + j][iz + k]);
00898         wmean += met0->w[ix + i][iy + j][iz + k];
00899         wsig += SQR(met0->w[ix + i][iy + j][iz + k]);
00900
00901         umean += met1->u[ix + i][iy + j][iz + k];
00902         usig += SQR(met1->u[ix + i][iy + j][iz + k]);
00903         vmean += met1->v[ix + i][iy + j][iz + k];
00904         vsig += SQR(met1->v[ix + i][iy + j][iz + k]);
00905         wmean += met1->w[ix + i][iy + j][iz + k];
00906         wsig += SQR(met1->w[ix + i][iy + j][iz + k]);
00907 #endif
00908     }
00909     usig = usig / 16.f - SQR(umean / 16.f);
00910     usig = (usig > 0 ? sqrtf(usig) : 0);
00911     vsig = vsig / 16.f - SQR(vmean / 16.f);
00912     vsig = (vsig > 0 ? sqrtf(vsig) : 0);
00913     wsig = wsig / 16.f - SQR(wmean / 16.f);
00914     wsig = (wsig > 0 ? sqrtf(wsig) : 0);
00915
00916     /* Set temporal correlations for mesoscale fluctuations... */
00917     double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00918     double r2 = sqrt(1 - r * r);
00919
00920     /* Calculate horizontal mesoscale wind fluctuations... */
00921     if (ctl->turb_mesox > 0) {
00922         cache->uvw[ip][0] =
00923             (float) (r * cache->uvw[ip][0] +
00924                 r2 * rs[3 * ip] * ctl->turb_mesox * usig);
00925         atm->lon[ip] +=
00926             DX2DEG(cache->uvw[ip][0] * dt[ip] / 1000., atm->lat[ip]);
00927
00928         cache->uvw[ip][1] =
00929             (float) (r * cache->uvw[ip][1] +
00930                 r2 * rs[3 * ip + 1] * ctl->turb_mesox * vsig);
00931         atm->lat[ip] += DY2DEG(cache->uvw[ip][1] * dt[ip] / 1000.);
00932     }
00933
00934     /* Calculate vertical mesoscale wind fluctuations... */
00935     if (ctl->turb_mesoz > 0) {
00936         cache->uvw[ip][2] =
00937             (float) (r * cache->uvw[ip][2] +
00938                 r2 * rs[3 * ip + 2] * ctl->turb_mesoz * wsig);
00939         atm->p[ip] += cache->uvw[ip][2] * dt[ip];
00940     }
00941 }
00942 }

```

Here is the call graph for this function:



5.47.2.6 module_diffusion_turb() void module_diffusion_turb (

```

    ctl_t * ctl,
    clim_t * clim,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate turbulent diffusion.

Definition at line 946 of file trac.c.

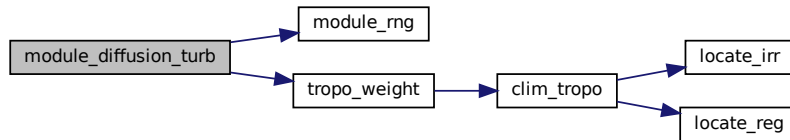
```

00951     {
00952
00953     /* Set timer... */
00954     SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);
00955
00956     /* Create random numbers... */
00957     module_rng(rs, 3 * (size_t) atm->np, 1);
00958
00959     const int np = atm->np;
00960 #ifdef _OPENACC
00961 #pragma acc data present(ctl,clim,atm,dt,rs)
00962 #pragma acc parallel loop independent gang vector
00963 #else
00964 #pragma omp parallel for default(shared)
00965 #endif
00966     for (int ip = 0; ip < np; ip++)
00967         if (dt[ip] != 0) {
00968
00969             /* Get weighting factor... */
00970             double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00971
00972             /* Set diffusivity... */
00973             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00974             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00975
00976             /* Horizontal turbulent diffusion... */
00977             if (dx > 0) {
00978                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00979                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00980                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00981             }
00982
00983             /* Vertical turbulent diffusion... */
00984             if (dz > 0) {
00985                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00986                 atm->p[ip] += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00987             }
00988         }

```

```
00989 }
```

Here is the call graph for this function:



5.47.2.7 module_dry_deposition() void module_dry_deposition (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate dry deposition.

Definition at line 993 of file [trac.c](#).

```

00998     {
00999
01000     /* Set timer... */
01001     SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);
01002
01003     /* Check quantity flags... */
01004     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01005         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01006
01007     const int np = atm->np;
01008 #ifdef _OPENACC
01009 #pragma acc data present(ctl, met0, met1, atm, dt)
01010 #pragma acc parallel loop independent gang vector
01011 #else
01012 #pragma omp parallel for default(shared)
01013 #endif
01014     for (int ip = 0; ip < np; ip++)
01015         if (dt[ip] != 0) {
01016             double ps, t, v_dep;
01017
01018             /* Get surface pressure... */
01019             INTPOL_INIT;
01020             INTPOL_2D(ps, 1);
01021
01022             /* Check whether particle is above the surface layer... */
01023             if (atm->p[ip] < ps - ctl->dry_depo_dp)
01024                 continue;
01025
01026             /* Set depth of surface layer... */
01027             double dz = 1000. * (Z(ps - ctl->dry_depo_dp) - Z(ps));
01028
01029             /* Calculate sedimentation velocity for particles... */
01030             if (ctl->qnt_rp > 0 && ctl->qnt_rhop > 0) {
01031
01032                 /* Get temperature... */
01033                 INTPOL_3D(t, 1);
01034
01035                 /* Set deposition velocity... */
01036                 v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01037                             atm->q[ctl->qnt_rhop][ip]);
01038             }
01039
01040             /* Use explicit sedimentation velocity for gases... */
01041             else

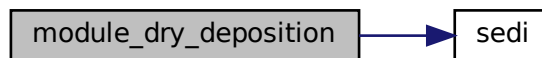
```

```

01043     v_dep = ctl->dry_depo_vdep;
01044
01045     /* Calculate loss of mass based on deposition velocity... */
01046     double aux = exp(-dt[ip] * v_dep / dz);
01047     if (ctl->qnt_m >= 0) {
01048         if (ctl->qnt_mloss_dry >= 0)
01049             atm->q[ctl->qnt_mloss_dry][ip]
01050                 += atm->q[ctl->qnt_m][ip] * (1 - aux);
01051         atm->q[ctl->qnt_m][ip] *= aux;
01052     }
01053     if (ctl->qnt_vmr >= 0)
01054         atm->q[ctl->qnt_vmr][ip] *= aux;
01055 }
01056 }

```

Here is the call graph for this function:



5.47.2.8 module_isosurf_init() void module_isosurf_init (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    cache_t * cache )

```

Initialize isosurface module.

Definition at line 1060 of file trac.c.

```

01065     {
01066
01067     FILE *in;
01068
01069     char line[LEN];
01070
01071     double t;
01072
01073     /* Set timer... */
01074     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01075
01076     /* Init... */
01077     INTPOL_INIT;
01078
01079     /* Save pressure... */
01080     if (ctl->isosurf == 1)
01081         for (int ip = 0; ip < atm->np; ip++)
01082             cache->iso_var[ip] = atm->p[ip];
01083
01084     /* Save density... */
01085     else if (ctl->isosurf == 2)
01086         for (int ip = 0; ip < atm->np; ip++) {
01087             INTPOL_3D(t, 1);
01088             cache->iso_var[ip] = atm->p[ip] / t;
01089         }
01090
01091     /* Save potential temperature... */
01092     else if (ctl->isosurf == 3)
01093         for (int ip = 0; ip < atm->np; ip++) {
01094             INTPOL_3D(t, 1);
01095             cache->iso_var[ip] = THETA(atm->p[ip], t);
01096         }

```

```

01097
01098  /* Read balloon pressure data... */
01099  else if (ctl->isosurf == 4) {
01100
01101      /* Write info... */
01102      LOG(1, "Read balloon pressure data: %s", ctl->balloon);
01103
01104      /* Open file... */
01105      if (!(in = fopen(ctl->balloon, "r")))
01106          ERRMSG("Cannot open file!");
01107
01108      /* Read pressure time series... */
01109      while (fgets(line, LEN, in))
01110          if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
01111                  &(cache->iso_ps[cache->iso_n])) == 2)
01112              if ((++cache->iso_n) > NP)
01113                  ERRMSG("Too many data points!");
01114
01115      /* Check number of points... */
01116      if (cache->iso_n < 1)
01117          ERRMSG("Could not read any data!");
01118
01119      /* Close file... */
01120      fclose(in);
01121  }
01122 }

```

5.47.2.9 module_isosurf() void module_isosurf (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    cache_t * cache )

```

Force air parcels to stay on isosurface.

Definition at line 1126 of file [trac.c](#).

```

01131     {
01132
01133     /* Set timer... */
01134     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01135
01136     const int np = atm->np;
01137     #ifdef _OPENACC
01138     #pragma acc data present(ctl, met0, met1, atm, cache)
01139     #pragma acc parallel loop independent gang vector
01140     #else
01141     #pragma omp parallel for default(shared)
01142     #endif
01143     for (int ip = 0; ip < np; ip++) {
01144
01145         double t;
01146
01147         /* Init... */
01148         INTPOL_INIT;
01149
01150         /* Restore pressure... */
01151         if (ctl->isosurf == 1)
01152             atm->p[ip] = cache->iso_var[ip];
01153
01154         /* Restore density... */
01155         else if (ctl->isosurf == 2) {
01156             INTPOL_3D(t, 1);
01157             atm->p[ip] = cache->iso_var[ip] * t;
01158         }
01159
01160         /* Restore potential temperature... */
01161         else if (ctl->isosurf == 3) {
01162             INTPOL_3D(t, 1);
01163             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
01164         }
01165
01166         /* Interpolate pressure... */
01167         else if (ctl->isosurf == 4) {
01168             if (atm->time[ip] <= cache->iso_ts[0])
01169                 atm->p[ip] = cache->iso_ps[0];
01170             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])

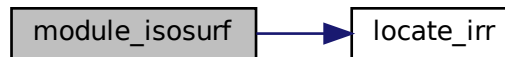
```

```

01171     atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
01172     else {
01173         int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
01174         atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
01175                        cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
01176                        atm->time[ip]);
01177     }
01178 }
01179 }
01180 }

```

Here is the call graph for this function:



5.47.2.10 module_meteo() void module_meteo (

```

    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm )

```

Interpolate meteo data for air parcel positions.

Definition at line 1184 of file [trac.c](#).

```

01189     {
01190
01191         /* Set timer... */
01192         SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);
01193
01194         /* Check quantity flags... */
01195         if (ctl->qnt_tsts >= 0)
01196             if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01197                 ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01198
01199         const int np = atm->np;
01200 #ifdef _OPENACC
01201 #pragma acc data present(ctl, clim, met0, met1, atm)
01202 #pragma acc parallel loop independent gang vector
01203 #else
01204 #pragma omp parallel for default(shared)
01205 #endif
01206         for (int ip = 0; ip < np; ip++) {
01207
01208             double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
01209                    cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
01210
01211             /* Interpolate meteo data... */
01212             INTPOL_INIT;
01213             INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
01214
01215             /* Set quantities... */
01216             SET_ATM(qnt_ps, ps);
01217             SET_ATM(qnt_ts, ts);
01218             SET_ATM(qnt_zs, zs);
01219             SET_ATM(qnt_us, us);
01220             SET_ATM(qnt_vs, vs);
01221             SET_ATM(qnt_pbl, pbl);
01222             SET_ATM(qnt_pt, pt);
01223             SET_ATM(qnt_tt, tt);
01224             SET_ATM(qnt_zt, zt);

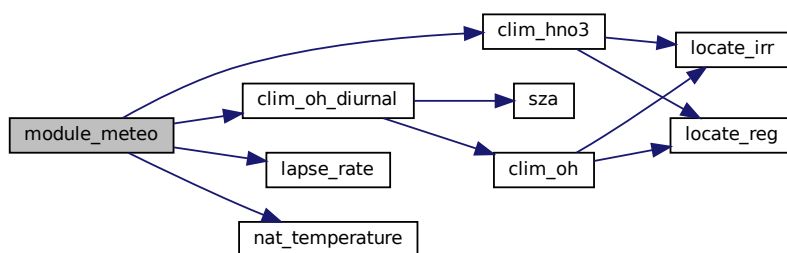
```

```

01225     SET_ATM(qnt_h2ot, h2ot);
01226     SET_ATM(qnt_z, z);
01227     SET_ATM(qnt_p, atm->p[ip]);
01228     SET_ATM(qnt_t, t);
01229     SET_ATM(qnt_rho, RHO(atm->p[ip], t));
01230     SET_ATM(qnt_u, u);
01231     SET_ATM(qnt_v, v);
01232     SET_ATM(qnt_w, w);
01233     SET_ATM(qnt_h2o, h2o);
01234     SET_ATM(qnt_o3, o3);
01235     SET_ATM(qnt_lwc, lwc);
01236     SET_ATM(qnt_iwc, iwc);
01237     SET_ATM(qnt_pct, pct);
01238     SET_ATM(qnt_pcb, pcb);
01239     SET_ATM(qnt_cl, cl);
01240     SET_ATM(qnt_plcl, plcl);
01241     SET_ATM(qnt_plfc, plfc);
01242     SET_ATM(qnt_pel, pel);
01243     SET_ATM(qnt_cape, cape);
01244     SET_ATM(qnt_cin, cin);
01245     SET_ATM(qnt_hno3,
01246             clim_hno3(clim, atm->time[ip], atm->lat[ip], atm->p[ip]));
01247     SET_ATM(qnt_oh,
01248             clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01249                             atm->lon[ip], atm->lat[ip]));
01250     SET_ATM(qnt_vh, sqrt(u * u + v * v));
01251     SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01252     SET_ATM(qnt_psat, PSAT(t));
01253     SET_ATM(qnt_psice, PSICE(t));
01254     SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
01255     SET_ATM(qnt_sh, SH(h2o));
01256     SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
01257     SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01258     SET_ATM(qnt_theta, THETA(atm->p[ip], t));
01259     SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
01260     SET_ATM(qnt_tvirt, TVIRT(t, h2o));
01261     SET_ATM(qnt_lapse, lapse_rate(t, h2o));
01262     SET_ATM(qnt_pv, pv);
01263     SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
01264     SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
01265     SET_ATM(qnt_tnat,
01266             nat_temperature(atm->p[ip], h2o,
01267                             clim_hno3(clim, atm->time[ip], atm->lat[ip],
01268                                     atm->p[ip])));
01269     SET_ATM(qnt_tsts,
01270             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01271 }
01272 }

```

Here is the call graph for this function:



```

5.47.2.11 module_oh_chem() void module_oh_chem (
    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,

```



```

    atm_t * atm,
    double * dt )

```

Calculate OH chemistry.

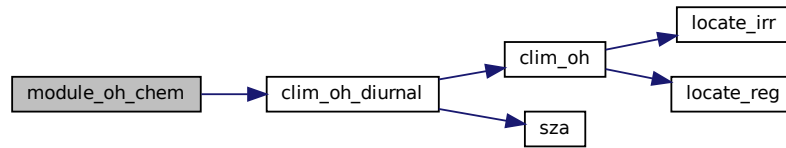
Definition at line 1276 of file trac.c.

```

01282     {
01283
01284     /* Set timer... */
01285     SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);
01286
01287     /* Check quantity flags... */
01288     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01289         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01290
01291     const int np = atm->np;
01292 #ifdef _OPENACC
01293 #pragma acc data present(ctl, clim, met0, met1, atm, dt)
01294 #pragma acc parallel loop independent gang vector
01295 #else
01296 #pragma omp parallel for default(shared)
01297 #endif
01298     for (int ip = 0; ip < np; ip++)
01299         if (dt[ip] != 0) {
01300
01301             /* Get temperature... */
01302             double t;
01303             INTPOL_INIT;
01304             INTPOL_3D(t, 1);
01305
01306             /* Use constant reaction rate... */
01307             double k = GSL_NAN;
01308             if (ctl->oh_chem_reaction == 1)
01309                 k = ctl->oh_chem[0];
01310
01311             /* Calculate bimolecular reaction rate... */
01312             else if (ctl->oh_chem_reaction == 2)
01313                 k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);
01314
01315             /* Calculate termolecular reaction rate... */
01316             if (ctl->oh_chem_reaction == 3) {
01317
01318                 /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
01319                 double M = 7.243e21 * (atm->p[ip] / 1000.) / t;
01320
01321                 /* Calculate rate coefficient for X + OH + M -> XOH + M
01322                  (JPL Publication 19-05) ... */
01323                 double k0 = ctl->oh_chem[0] *
01324                     (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
01325                 double ki = ctl->oh_chem[2] *
01326                     (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
01327                 double c = log10(k0 * M / ki);
01328                 k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01329             }
01330
01331             /* Calculate exponential decay... */
01332             double rate_coef =
01333                 k * clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01334                                     atm->lon[ip],
01335                                     atm->lat[ip]);
01336             double aux = exp(-dt[ip] * rate_coef);
01337             if (ctl->qnt_m >= 0) {
01338                 if (ctl->qnt_mloss_oh >= 0)
01339                     atm->q[ctl->qnt_mloss_oh][ip]
01340                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01341                 atm->q[ctl->qnt_m][ip] *= aux;
01342             }
01343             if (ctl->qnt_vmr >= 0)
01344                 atm->q[ctl->qnt_vmr][ip] *= aux;
01345         }
01346     }

```

Here is the call graph for this function:



5.47.2.12 module_h2o2_chem() void module_h2o2_chem (

```

    ctl_t * ctl,
    clim_t * clim,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt,
    double * rs )

```

Calculate H2O2 chemistry.

Definition at line 1350 of file trac.c.

```

01357     {
01358
01359     /* Set timer... */
01360     SELECT_TIMER("MODULE_H2O2CHEM", "PHYSICS", NVTX_GPU);
01361
01362     /* Check quantity flags... */
01363     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01364         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01365     if (ctl->qnt_vmrimpl < 0)
01366         ERRMSG("Module needs quantity implicit volume mixing ratio!");
01367
01368     /* Create random numbers... */
01369     module_rng(rs, (size_t) atm->np, 0);
01370
01371     const int np = atm->np;
01372 #ifndef _OPENACC
01373 #pragma acc data present(clim,ctl,met0,met1,atm,dt,rs)
01374 #pragma acc parallel loop independent gang vector
01375 #else
01376 #pragma omp parallel for default(shared)
01377 #endif
01378     for (int ip = 0; ip < np; ip++)
01379         if (dt[ip] != 0) {
01380
01381             /* Check whether particle is inside cloud... */
01382             double lwc;
01383             INTPOL_INIT;
01384             INTPOL_3D(lwc, 1);
01385             if (!(lwc > 0))
01386                 continue;
01387
01388             /* Check cloud cover... */
01389             if (rs[ip] > ctl->h2o2_chem_cc)
01390                 continue;
01391
01392             /* Check implicit volume mixing ratio... */
01393             if (atm->q[ctl->qnt_vmrimpl][ip] == 0)
01394                 continue;
01395
01396             /* Get temperature... */
01397             double t;
01398             INTPOL_3D(t, 0);
01399
01400             /* Reaction rate (Berglen et al., 2004)... */

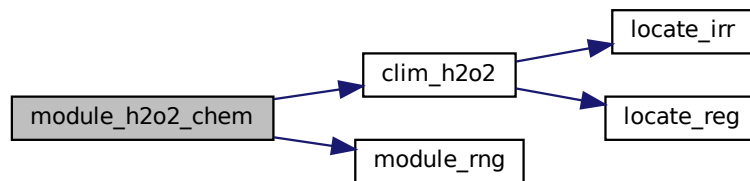
```

```

01401     double k = 9.1e7 * exp(-29700 / RI * (1. / t - 1. / 298.15)); // Maass 1999 unit: M^(-2)
01402
01403     /* Henry constant of SO2... */
01404     double H_SO2 = 1.3e-2 * exp(2900 * (1. / t - 1. / 298.15)) * RI * t;
01405     double K_1S = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15)); // unit: M
01406
01407     /* Henry constant of H2O2... */
01408     double H_h2o2 = 8.3e2 * exp(7600 * (1 / t - 1 / 298.15)) * RI * t;
01409
01410     /* Concentration of H2O2 (Barth et al., 1989)... */
01411     double SO2 = atm->q[ctl->qnt_vmrimpl][ip] * 1e9; // vmr unit: ppbv
01412     double h2o2 = H_h2o2
01413         * clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip])
01414         * 0.59 * exp(-0.687 * SO2) * 1000 / 6.02214e23; // unit: M
01415
01416     /* Volume water content in cloud [m^3 m^(-3)]... */
01417     double rho_air = 100 * atm->p[ip] / (RA * t);
01418     double CWC = lwc * rho_air / 1000;
01419
01420     /* Calculate exponential decay (Rolph et al., 1992)... */
01421     double rate_coef = k * K_1S * h2o2 * H_SO2 * CWC;
01422     double aux = exp(-dt[ip] * rate_coef);
01423     if (ctl->qnt_m >= 0) {
01424         if (ctl->qnt_mloss_h2o2 >= 0)
01425             atm->q[ctl->qnt_mloss_h2o2][ip] +=
01426                 atm->q[ctl->qnt_m][ip] * (1 - aux);
01427         atm->q[ctl->qnt_m][ip] *= aux;
01428     }
01429     if (ctl->qnt_vmr >= 0)
01430         atm->q[ctl->qnt_vmr][ip] *= aux;
01431 }
01432 }

```

Here is the call graph for this function:



5.47.2.13 module_chemgrid() void module_chemgrid (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Interpolate to chemistry grid.

Definition at line 1436 of file trac.c.

```

01441     {
01442
01443     double *mass, *z, *lon, *lat, *press, *area;
01444
01445     int *ixs, *iys, *izs;
01446
01447     /* Update host... */
01448     #ifdef _OPENACC
01449     SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
01450     #pragma acc update host(atm[:1])

```

```

01451 #endif
01452
01453 /* Set timer... */
01454 SELECT_TIMER("MODULE_CHEMGRID", "PHYSICS", NVTX_GPU);
01455
01456 /* Check quantity flags... */
01457 if (ctl->qnt_m < 0)
01458     ERRMSG("Module needs quantity mass!");
01459 if (ctl->qnt_vmrimpl < 0)
01460     ERRMSG("Module needs quantity implicit volume mixing ratio!");
01461
01462 /* Allocate... */
01463 ALLOC(mass, double,
01464       ctl->chemgrid_nx * ctl->chemgrid_ny * ctl->chemgrid_nz);
01465 ALLOC(z, double,
01466       ctl->chemgrid_nz);
01467 ALLOC(lon, double,
01468       ctl->chemgrid_nx);
01469 ALLOC(lat, double,
01470       ctl->chemgrid_ny);
01471 ALLOC(area, double,
01472       ctl->chemgrid_ny);
01473 ALLOC(press, double,
01474       ctl->chemgrid_nz);
01475 ALLOC(ixs, int,
01476       atm->np);
01477 ALLOC(iys, int,
01478       atm->np);
01479 ALLOC(izs, int,
01480       atm->np);
01481
01482 /* Set grid box size... */
01483 double dz = (ctl->chemgrid_z1 - ctl->chemgrid_z0) / ctl->chemgrid_nz;
01484 double dlon = (ctl->chemgrid_lon1 - ctl->chemgrid_lon0) / ctl->chemgrid_nx;
01485 double dlat = (ctl->chemgrid_lat1 - ctl->chemgrid_lat0) / ctl->chemgrid_ny;
01486
01487 /* Set vertical coordinates... */
01488 #pragma omp parallel for default(shared)
01489 for (int iz = 0; iz < ctl->chemgrid_nz; iz++) {
01490     z[iz] = ctl->chemgrid_z0 + dz * (iz + 0.5);
01491     press[iz] = P(z[iz]);
01492 }
01493
01494 /* Set horizontal coordinates... */
01495 for (int ix = 0; ix < ctl->chemgrid_nx; ix++)
01496     lon[ix] = ctl->chemgrid_lon0 + dlon * (ix + 0.5);
01497 #pragma omp parallel for default(shared)
01498 for (int iy = 0; iy < ctl->chemgrid_ny; iy++) {
01499     lat[iy] = ctl->chemgrid_lat0 + dlat * (iy + 0.5);
01500     area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
01501               * cos(lat[iy] * M_PI / 180.);
01502 }
01503
01504 /* Set time interval for output... */
01505 double t0 = t - 0.5 * ctl->dt_mod;
01506 double t1 = t + 0.5 * ctl->dt_mod;
01507
01508 /* Get indices... */
01509 #pragma omp parallel for default(shared)
01510 for (int ip = 0; ip < atm->np; ip++) {
01511     ixs[ip] = (int) ((atm->lon[ip] - ctl->chemgrid_lon0) / dlon);
01512     iys[ip] = (int) ((atm->lat[ip] - ctl->chemgrid_lat0) / dlat);
01513     izs[ip] = (int) ((Z(atm->p[ip]) - ctl->chemgrid_z0) / dz);
01514     if (atm->time[ip] < t0 || atm->time[ip] > t1
01515         || ixs[ip] < 0 || ixs[ip] >= ctl->chemgrid_nx
01516         || iys[ip] < 0 || iys[ip] >= ctl->chemgrid_ny
01517         || izs[ip] < 0 || izs[ip] >= ctl->chemgrid_nz)
01518         izs[ip] = -1;
01519 }
01520
01521 /* Average data... */
01522 for (int ip = 0; ip < atm->np; ip++)
01523     if (izs[ip] >= 0)
01524         mass[ARRAY_3D
01525              (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
01526             += atm->q[ctl->qnt_m][ip];
01527
01528 /* Interpolate volume mixing ratio... */
01529 #pragma omp parallel for default(shared)
01530 for (int ip = 0; ip < atm->np; ip++)
01531     if (izs[ip] >= 0) {
01532         double temp;
01533         INTPOL_INIT;
01534         intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
01535                            lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
01536         atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass *
01537             mass[ARRAY_3D

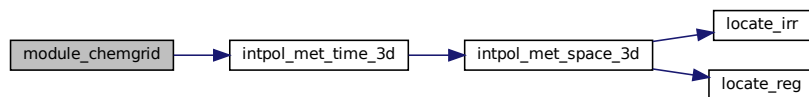
```

```

01538         (ixs[ip], iys[ip], ctl->chemgrid_ny, ize[ip], ctl->chemgrid_nz])
01539         / (RHO(press[ize[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
01540     }
01541
01542     /* Free... */
01543     free(mass);
01544     free(z);
01545     free(lon);
01546     free(lat);
01547     free(area);
01548     free(press);
01549     free(ixs);
01550     free(iys);
01551     free(ize);
01552
01553     /* Update device... */
01554     #ifdef _OPENACC
01555         SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
01556     #pragma acc update device(atm[:1])
01557     #endif
01558 }

```

Here is the call graph for this function:



5.47.2.14 module_position() void module_position (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Check position of air parcels.

Definition at line 1562 of file [trac.c](#).

```

01567     {
01568
01569         /* Set timer... */
01570         SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);
01571
01572         const int np = atm->np;
01573         #ifdef _OPENACC
01574         #pragma acc data present(met0, met1, atm, dt)
01575         #pragma acc parallel loop independent gang vector
01576         #else
01577         #pragma omp parallel for default(shared)
01578         #endif
01579         for (int ip = 0; ip < np; ip++)
01580             if (dt[ip] != 0) {
01581
01582                 /* Init... */
01583                 double ps;
01584                 INTPOL_INIT;
01585
01586                 /* Calculate modulo... */
01587                 atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01588                 atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01589
01590                 /* Check latitude... */
01591                 while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01592                     if (atm->lat[ip] > 90) {
01593                         atm->lat[ip] = 180 - atm->lat[ip];

```

```

01594         atm->lon[ip] += 180;
01595     }
01596     if (atm->lat[ip] < -90) {
01597         atm->lat[ip] = -180 - atm->lat[ip];
01598         atm->lon[ip] += 180;
01599     }
01600 }
01601
01602 /* Check longitude... */
01603 while (atm->lon[ip] < -180)
01604     atm->lon[ip] += 360;
01605 while (atm->lon[ip] >= 180)
01606     atm->lon[ip] -= 360;
01607
01608 /* Check pressure... */
01609 if (atm->p[ip] < met0->p[met0->np - 1]) {
01610     if (ctl->reflect)
01611         atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
01612     else
01613         atm->p[ip] = met0->p[met0->np - 1];
01614 } else if (atm->p[ip] > 300.) {
01615     INTPOL_2D(ps, 1);
01616     if (atm->p[ip] > ps) {
01617         if (ctl->reflect)
01618             atm->p[ip] = 2. * ps - atm->p[ip];
01619         else
01620             atm->p[ip] = ps;
01621     }
01622 }
01623 }
01624 }

```

5.47.2.15 module_rng_init() void module_rng_init (
int ntask)

Initialize random number generator...

Definition at line 1628 of file [trac.c](#).

```

01629 {
01630
01631     /* Initialize random number generator... */
01632 #ifdef _OPENACC
01633
01634     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT) !=
01635         CURAND_STATUS_SUCCESS)
01636         ERRMSG("Cannot create random number generator!");
01637     if (curandSetPseudoRandomGeneratorSeed(rng, ntask) != CURAND_STATUS_SUCCESS)
01638         ERRMSG("Cannot set seed for random number generator!");
01639     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01640         != CURAND_STATUS_SUCCESS)
01641         ERRMSG("Cannot set stream for random number generator!");
01642
01643 #else
01644     gsl_rng_env_setup();
01645     if (omp_get_max_threads() > NTHREADS)
01646         ERRMSG("Too many threads!");
01647     for (int i = 0; i < NTHREADS; i++) {
01648         rng[i] = gsl_rng_alloc(gsl_rng_default);
01649         gsl_rng_set(rng[i],
01650             gsl_rng_default_seed + (long unsigned) (ntask * NTHREADS +
01651                 i));
01652     }
01653 }
01654 #endif
01655 }
01656 }

```

5.47.2.16 module_rng() void module_rng (
double * rs,
size_t n,
int method)

Generate random numbers.

Definition at line 1660 of file [trac.c](#).

```

01663         {
01664
01665 #ifdef _OPENACC
01666
01667 #pragma acc host_data use_device(rs)
01668 {
01669     /* Uniform distribution... */
01670     if (method == 0) {
01671         if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n)) !=
01672             CURAND_STATUS_SUCCESS)
01673             ERRMSG("Cannot create random numbers!");
01674     }
01675
01676     /* Normal distribution... */
01677     else if (method == 1) {
01678         if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0) !=
01679             CURAND_STATUS_SUCCESS)
01680             ERRMSG("Cannot create random numbers!");
01681     }
01682 }
01683
01684 #else
01685
01686     /* Uniform distribution... */
01687     if (method == 0) {
01688 #pragma omp parallel for default(shared)
01689         for (size_t i = 0; i < n; ++i)
01690             rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01691     }
01692
01693     /* Normal distribution... */
01694     else if (method == 1) {
01695 #pragma omp parallel for default(shared)
01696         for (size_t i = 0; i < n; ++i)
01697             rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01698     }
01699 #endif
01700 }
```

5.47.2.17 module_sedi() void module_sedi (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )
```

Calculate sedimentation of air parcels.

Definition at line 1704 of file [trac.c](#).

```

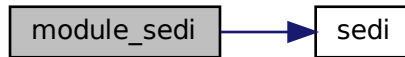
01709         {
01710
01711     /* Set timer... */
01712     SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);
01713
01714     const int np = atm->np;
01715 #ifdef _OPENACC
01716 #pragma acc data present(ctl, met0, met1, atm, dt)
01717 #pragma acc parallel loop independent gang vector
01718 #else
01719 #pragma omp parallel for default(shared)
01720 #endif
01721     for (int ip = 0; ip < np; ip++)
01722         if (dt[ip] != 0) {
01723
01724             /* Get temperature... */
01725             double t;
01726             INTPOL_INIT;
01727             INTPOL_3D(t, 1);
01728
01729             /* Sedimentation velocity... */
01730             double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01731                             atm->q[ctl->qnt_rhop][ip]);
01732         }
```

```

01733      /* Calculate pressure change... */
01734      atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01735  }
01736 }

```

Here is the call graph for this function:



5.47.2.18 module_sort() void module_sort (

```

    ctl_t * ctl,
    met_t * met0,
    atm_t * atm )

```

Sort particles according to box index.

Definition at line 1740 of file [trac.c](#).

```

01743 {
01744
01745     /* Set timer... */
01746     SELECT_TIMER("MODULE_SORT_BOXINDEX", "MEMORY", NVTX_GPU);
01747
01748     /* Allocate... */
01749     const int np = atm->np;
01750     double *restrict const a = (double *) malloc((size_t) np * sizeof(double));
01751     int *restrict const p = (int *) malloc((size_t) np * sizeof(int));
01752
01753     #ifdef _OPENACC
01754     #pragma acc enter data create(a[0:np],p[0:np])
01755     #pragma acc data present(ctl,met0,atm,a,p)
01756     #endif
01757
01758     /* Get box index... */
01759     #ifdef _OPENACC
01760     #pragma acc parallel loop independent gang vector
01761     #else
01762     #pragma omp parallel for default(shared)
01763     #endif
01764     for (int ip = 0; ip < np; ip++) {
01765         a[ip] =
01766             (double) ((locate_reg(met0->lon, met0->nx, atm->lon[ip]) * met0->ny +
01767                 locate_reg(met0->lat, met0->ny,
01768                     atm->lat[ip])) * met0->np + locate_irr(met0->p,
01769                     met0->np,
01770                     atm->p,
01771                     [ip]));
01772         p[ip] = ip;
01773     }
01774
01775     /* Set timer... */
01776     SELECT_TIMER("MODULE_SORT_THRUST", "MEMORY", NVTX_GPU);
01777
01778     /* Sorting... */
01779     #ifdef _OPENACC
01780     {
01781         #ifdef THRUST
01782         {
01783             #pragma acc host_data use_device(a, p)
01784             thrustSortWrapper(a, np, p);
01785         }
01786     #else
01787     {

```

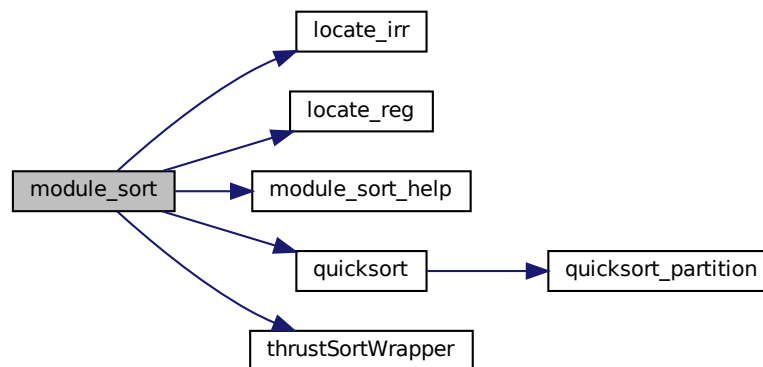


```

01788 #pragma acc update host(a[0:np], p[0:np])
01789 #pragma omp parallel
01790 {
01791 #pragma omp single nowait
01792     quicksort(a, p, 0, np - 1);
01793 }
01794 #pragma acc update device(a[0:np], p[0:np])
01795 }
01796 #endif
01797 }
01798 #else
01799 {
01800 #ifdef THRUST
01801     {
01802         thrustSortWrapper(a, np, p);
01803     }
01804 #else
01805 {
01806 #pragma omp parallel
01807 {
01808 #pragma omp single nowait
01809     quicksort(a, p, 0, np - 1);
01810 }
01811 }
01812 #endif
01813 }
01814 #endif
01815
01816 /* Set timer... */
01817 SELECT_TIMER("MODULE_SORT_REORDERING", "MEMORY", NVTX_GPU);
01818
01819 /* Sort data... */
01820 module_sort_help(atm->time, p, np);
01821 module_sort_help(atm->p, p, np);
01822 module_sort_help(atm->lon, p, np);
01823 module_sort_help(atm->lat, p, np);
01824 for (int iq = 0; iq < ctl->nq; iq++)
01825     module_sort_help(atm->q[iq], p, np);
01826
01827 /* Free... */
01828 #ifdef _OPENACC
01829 #pragma acc exit data delete(a,p)
01830 #endif
01831 free(a);
01832 free(p);
01833 }

```

Here is the call graph for this function:



5.47.2.19 module_sort_help() void module_sort_help (
double * a,

```

int * p,
int np )

```

Helper function for sorting module.

Definition at line 1837 of file [trac.c](#).

```

01840     {
01841
01842     /* Allocate... */
01843     double *restrict const help =
01844         (double *) malloc((size_t) np * sizeof(double));
01845
01846     /* Reordering of array... */
01847     #ifdef _OPENACC
01848     #pragma acc enter data create(help[0:np])
01849     #pragma acc data present(a,p,help)
01850     #pragma acc parallel loop independent gang vector
01851     #endif
01852     for (int ip = 0; ip < np; ip++)
01853         help[ip] = a[p[ip]];
01854     #ifdef _OPENACC
01855     #pragma acc parallel loop independent gang vector
01856     #endif
01857     for (int ip = 0; ip < np; ip++)
01858         a[ip] = help[ip];
01859
01860     /* Free... */
01861     #ifdef _OPENACC
01862     #pragma acc exit data delete(help)
01863     #endif
01864     free(help);
01865 }

```

5.47.2.20 module_timesteps() void module_timesteps (

```

    ctl_t * ctl,
    atm_t * atm,
    met_t * met0,
    double * dt,
    double t )

```

Calculate time steps.

Definition at line 1869 of file [trac.c](#).

```

01874     {
01875
01876     /* Set timer... */
01877     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01878
01879     const double latmin = gsl_stats_min(met0->lat, 1, (size_t) met0->ny),
01880         latmax = gsl_stats_max(met0->lat, 1, (size_t) met0->ny);
01881
01882     const int np = atm->np,
01883         local = (fabs(met0->lon[met0->nx - 1] - met0->lon[0] - 360.0) >= 0.01);
01884
01885     #ifdef _OPENACC
01886     #pragma acc data present(ctl, atm, dt)
01887     #pragma acc parallel loop independent gang vector
01888     #else
01889     #pragma omp parallel for default(shared)
01890     #endif
01891     for (int ip = 0; ip < np; ip++) {
01892
01893         /* Set time step for each air parcel... */
01894         if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
01895             && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
01896             && ctl->direction * (atm->time[ip] - t) < 0))
01897             dt[ip] = t - atm->time[ip];
01898         else
01899             dt[ip] = 0.0;
01900
01901         /* Check horizontal boundaries of local meteo data... */
01902         if (local && (atm->lon[ip] <= met0->lon[0]
01903             || atm->lon[ip] >= met0->lon[met0->nx - 1]
01904             || atm->lat[ip] <= latmin || atm->lat[ip] >= latmax))
01905             dt[ip] = 0.0;
01906     }
01907 }

```

5.47.2.21 module_timesteps_init() void module_timesteps_init (

```

    ctl_t * ctl,
    atm_t * atm )

```

Initialize timesteps.

Definition at line 1911 of file [trac.c](#).

```

01913     {
01914
01915     /* Set timer... */
01916     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01917
01918     /* Set start time... */
01919     if (ctl->direction == 1) {
01920         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01921         if (ctl->t_stop > 1e99)
01922             ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01923     } else {
01924         ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01925         if (ctl->t_stop > 1e99)
01926             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01927     }
01928
01929     /* Check time interval... */
01930     if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
01931         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
01932
01933     /* Round start time... */
01934     if (ctl->direction == 1)
01935         ctl->t_start = floor(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01936     else
01937         ctl->t_start = ceil(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01938 }

```

5.47.2.22 module_wet_deposition() void module_wet_deposition (

```

    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double * dt )

```

Calculate wet deposition.

Definition at line 1942 of file [trac.c](#).

```

01947     {
01948
01949     /* Set timer... */
01950     SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);
01951
01952     /* Check quantity flags... */
01953     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01954         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01955
01956     const int np = atm->np;
01957     #ifdef _OPENACC
01958     #pragma acc data present(ctl, met0, met1, atm, dt)
01959     #pragma acc parallel loop independent gang vector
01960     #else
01961     #pragma omp parallel for default(shared)
01962     #endif
01963     for (int ip = 0; ip < np; ip++)
01964         if (dt[ip] != 0) {
01965
01966             double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;
01967
01968             /* Check whether particle is below cloud top... */
01969             INTPOL_INIT;
01970             INTPOL_2D(pct, 1);
01971             if (!isfinite(pct) || atm->p[ip] <= pct)
01972                 continue;
01973
01974             /* Get cloud bottom pressure... */
01975             INTPOL_2D(pcb, 0);
01976

```

```

01977      /* Estimate precipitation rate (Pisso et al., 2019)... */
01978      INTPOL_2D(cl, 0);
01979      double Is =
01980          pow(1. / ctl->wet_depo_pre[0] * cl, 1. / ctl->wet_depo_pre[1]);
01981      if (Is < 0.01)
01982          continue;
01983
01984      /* Check whether particle is inside or below cloud... */
01985      INTPOL_3D(lwc, 1);
01986      INTPOL_3D(iwc, 0);
01987      int inside = (iwc > 0 || lwc > 0);
01988
01989      /* Get temperature... */
01990      INTPOL_3D(t, 0);
01991
01992      /* Calculate in-cloud scavenging coefficient... */
01993      if (inside) {
01994
01995          /* Calculate retention factor... */
01996          double eta;
01997          if (t > 273.15)
01998              eta = 1;
01999          else if (t <= 238.15)
02000              eta = ctl->wet_depo_ic_ret_ratio;
02001          else
02002              eta = LIN(273.15, 1, 238.15, ctl->wet_depo_ic_ret_ratio, t);
02003
02004          /* Use exponential dependency for particles ... */
02005          if (ctl->wet_depo_ic_a > 0)
02006              lambda = ctl->wet_depo_ic_a * pow(Is, ctl->wet_depo_ic_b) * eta;
02007
02008          /* Use Henry's law for gases... */
02009          else if (ctl->wet_depo_ic_h[0] > 0) {
02010
02011              /* Get Henry's constant (Sander, 2015)... */
02012              h = ctl->wet_depo_ic_h[0]
02013                  * exp(ctl->wet_depo_ic_h[1] * (1. / t - 1. / 298.15));
02014
02015              /* Use effective Henry's constant for SO2
02016               (Berglen, 2004; Simpson, 2012)... */
02017              if (ctl->wet_depo_ic_h[2] > 0) {
02018                  double H_ion = pow(10, ctl->wet_depo_ic_h[2] * (-1));
02019                  double K_1 = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));
02020                  double K_2 = 6e-8 * exp(1.12e3 * (1. / t - 1. / 298.15));
02021                  h *= (1 + K_1 / H_ion + K_1 * K_2 / pow(H_ion, 2));
02022              }
02023
02024              /* Estimate depth of cloud layer... */
02025              dz = 1e3 * (Z(pct) - Z(pcb));
02026
02027              /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
02028              lambda = h * RI * t * Is / 3.6e6 / dz * eta;
02029          }
02030      }
02031
02032      /* Calculate below-cloud scavenging coefficient... */
02033      else {
02034
02035          /* Calculate retention factor... */
02036          double eta;
02037          if (t > 270)
02038              eta = 1;
02039          else
02040              eta = ctl->wet_depo_bc_ret_ratio;
02041
02042          /* Use exponential dependency for particles... */
02043          if (ctl->wet_depo_bc_a > 0)
02044              lambda = ctl->wet_depo_bc_a * pow(Is, ctl->wet_depo_bc_b) * eta;
02045
02046          /* Use Henry's law for gases... */
02047          else if (ctl->wet_depo_bc_h[0] > 0) {
02048
02049              /* Get Henry's constant (Sander, 2015)... */
02050              h = ctl->wet_depo_bc_h[0]
02051                  * exp(ctl->wet_depo_bc_h[1] * (1. / t - 1. / 298.15));
02052
02053              /* Estimate depth of cloud layer... */
02054              dz = 1e3 * (Z(pct) - Z(pcb));
02055
02056              /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
02057              lambda = h * RI * t * Is / 3.6e6 / dz * eta;
02058          }
02059      }
02060
02061      /* Calculate exponential decay of mass... */
02062      double aux = exp(-dt[ip] * lambda);
02063      if (ctl->qnt_m >= 0) {

```

```

02064         if (ctl->qnt_mloss_wet >= 0)
02065             atm->q[ctl->qnt_mloss_wet][ip]
02066             += atm->q[ctl->qnt_m][ip] * (1 - aux);
02067         atm->q[ctl->qnt_m][ip] *= aux;
02068     }
02069     if (ctl->qnt_vmr >= 0)
02070         atm->q[ctl->qnt_vmr][ip] *= aux;
02071     }
02072 }

```

5.47.2.23 write_output() void write_output (

```

    const char * dirname,
    ctl_t * ctl,
    met_t * met0,
    met_t * met1,
    atm_t * atm,
    double t )

```

Write simulation output.

Definition at line 2076 of file trac.c.

```

02082     {
02083
02084         char ext[10], filename[2 * LEN];
02085
02086         double r;
02087
02088         int year, mon, day, hour, min, sec;
02089
02090         /* Get time... */
02091         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02092
02093         /* Update host... */
02094         #ifdef _OPENACC
02095         if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
02096             || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
02097             || (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0)
02098             || (ctl->csi_basename[0] != '-' || ctl->prof_basename[0] != '-')
02099             || (ctl->sample_basename[0] != '-' || ctl->stat_basename[0] != '-')) {
02100             SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
02101             #pragma acc update host(atm[:1])
02102         }
02103         #endif
02104
02105         /* Write atmospheric data... */
02106         if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
02107             if (ctl->atm_type == 0)
02108                 sprintf(ext, "tab");
02109             else if (ctl->atm_type == 1)
02110                 sprintf(ext, "bin");
02111             else if (ctl->atm_type == 2)
02112                 sprintf(ext, "nc");
02113             sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.%s",
02114                     dirname, ctl->atm_basename, year, mon, day, hour, min, ext);
02115             write_atm(filename, ctl, atm, t);
02116         }
02117
02118         /* Write gridded data... */
02119         if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
02120             sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.%s",
02121                     dirname, ctl->grid_basename, year, mon, day, hour, min,
02122                     ctl->grid_type == 0 ? "tab" : "nc");
02123             write_grid(filename, ctl, met0, met1, atm, t);
02124         }
02125
02126         /* Write CSI data... */
02127         if (ctl->csi_basename[0] != '-') {
02128             sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
02129             write_csi(filename, ctl, atm, t);
02130         }
02131
02132         /* Write ensemble data... */
02133         if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
02134             sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
02135                     dirname, ctl->ens_basename, year, mon, day, hour, min);
02136             write_ens(filename, ctl, atm, t);
02137         }
02138     }

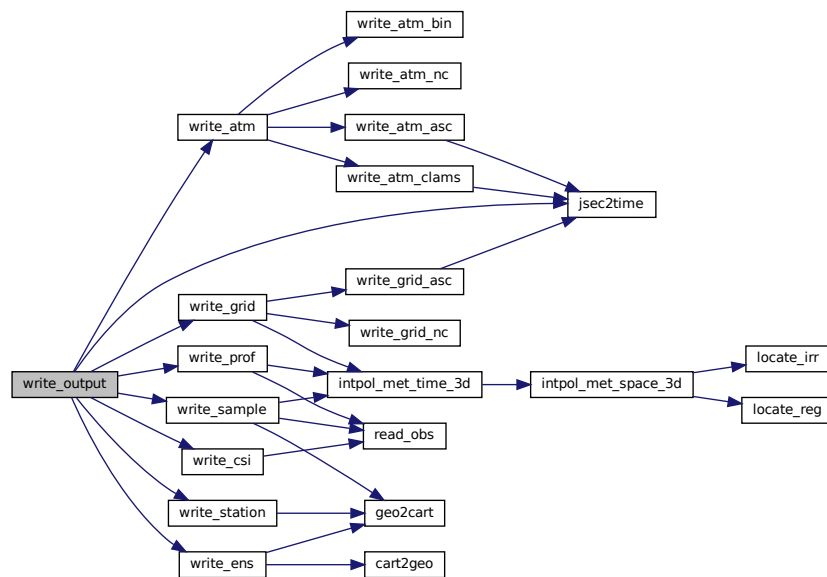
```

```

02137     }
02138
02139     /* Write profile data... */
02140     if (ctl->prof_basename[0] != '-') {
02141         sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
02142         write_prof(filename, ctl, met0, met1, atm, t);
02143     }
02144
02145     /* Write sample data... */
02146     if (ctl->sample_basename[0] != '-') {
02147         sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
02148         write_sample(filename, ctl, met0, met1, atm, t);
02149     }
02150
02151     /* Write station data... */
02152     if (ctl->stat_basename[0] != '-') {
02153         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
02154         write_station(filename, ctl, atm, t);
02155     }
02156 }

```

Here is the call graph for this function:



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

Definition at line 222 of file [trac.c](#).

```

00224     {
00225
00226         ctl_t ctl;
00227
00228         atm_t *atm;
00229
00230         cache_t *cache;
00231
00232         clim_t *clim;
00233
00234         met_t *met0, *met1;
00235
00236 #ifdef ASYNCIO
00237         met_t *met0TMP, *met1TMP, *mets;

```

```

00238     ctl_t   ctlTMP;
00239 #endif
00240
00241     FILE *dirlist;
00242
00243     char dirname[LEN], filename[2 * LEN];
00244
00245     double *dt, *rs, t;
00246
00247     int ntask = -1, rank = 0, size = 1;
00248
00249     /* Start timers... */
00250     START_TIMERS;
00251
00252     /* Initialize MPI... */
00253 #ifdef MPI
00254     SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
00255     MPI_Init(&argc, &argv);
00256     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00257     MPI_Comm_size(MPI_COMM_WORLD, &size);
00258 #endif
00259
00260     /* Initialize GPUs... */
00261 #ifdef _OPENACC
00262     SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
00263     if (acc_get_num_devices(acc_device_nvidia) <= 0)
00264         ERRMSG("Not running on a GPU device!");
00265     acc_set_device_num(rank % acc_get_num_devices(acc_device_nvidia),
00266                       acc_device_nvidia);
00267     acc_device_t device_type = acc_get_device_type();
00268     acc_init(device_type);
00269 #endif
00270
00271     /* Check arguments... */
00272     if (argc < 4)
00273         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00274
00275     /* Open directory list... */
00276     if (!(dirlist = fopen(argv[1], "r")))
00277         ERRMSG("Cannot open directory list!");
00278
00279     /* Loop over directories... */
00280     while (fscanf(dirlist, "%4999s", dirname) != EOF) {
00281
00282         /* MPI parallelization... */
00283         if ((++ntask) % size != rank)
00284             continue;
00285
00286         /* -----
00287            Initialize model run...
00288            ----- */
00289
00290         /* Allocate... */
00291         SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
00292         ALLOC(atm, atm_t, 1);
00293         ALLOC(cache, cache_t, 1);
00294         ALLOC(clim, clim_t, 1);
00295         ALLOC(met0, met_t, 1);
00296         ALLOC(met1, met_t, 1);
00297 #ifdef ASYNCIO
00298         ALLOC(met0TMP, met_t, 1);
00299         ALLOC(met1TMP, met_t, 1);
00300 #endif
00301         ALLOC(dt, double,
00302              NP);
00303         ALLOC(rs, double,
00304              3 * NP + 1);
00305
00306         /* Create data region on GPUs... */
00307 #ifdef _OPENACC
00308         SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
00309 #ifdef ASYNCIO
00310 #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl,ctlTMP, met0[:1], met1[:1],
00311                               met0TMP[:1], met1TMP[:1], dt[:NP], rs[:3 * NP])
00312 #else
00313 #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3 *
00314                               NP])
00315 #endif
00316 #endif
00317
00318         /* Read control parameters... */
00319         sprintf(filename, "%s/%s", dirname, argv[2]);
00320         read_ctl(filename, argc, argv, &ctl);
00321
00322         /* Read climatological data... */
00323         read_clim(&ctl, clim);
00324

```

```

00323      /* Read atmospheric data... */
00324      sprintf(filename, "%s/%s", dirname, argv[3]);
00325      if (!read_atm(filename, &ctl, atm))
00326          ERRMSG("Cannot open file!");
00327
00328      /* Initialize timesteps... */
00329      module_timesteps_init(&ctl, atm);
00330
00331      /* Update GPU... */
00332      #ifdef _OPENACC
00333          SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00334          #pragma acc update device(atm[:1], clim[:1], ctl)
00335      #endif
00336
00337      /* Initialize random number generator... */
00338      module_rng_init(ntask);
00339
00340      /* Initialize meteo data... */
00341      #ifdef ASYNCIO
00342          ctlTMP = ctl;
00343      #endif
00344      get_met(&ctl, clim, ctl.t_start, &met0, &met1);
00345      if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00346          WARN("Violation of CFL criterion! Check DT_MOD!");
00347      #ifdef ASYNCIO
00348          get_met(&ctlTMP, clim, ctlTMP.t_start, &met0TMP, &met1TMP);
00349      #endif
00350
00351      /* Initialize isosurface... */
00352      if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00353          module_isosurf_init(&ctl, met0, met1, atm, cache);
00354
00355      /* Update GPU... */
00356      #ifdef _OPENACC
00357          SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00358          #pragma acc update device(cache[:1])
00359      #endif
00360
00361      /* -----
00362      Loop over timesteps...
00363      ----- */
00364
00365      /* Loop over timesteps... */
00366      #ifdef ASYNCIO
00367          omp_set_nested(1);
00368          // omp_set_dynamic(0);
00369          int ompTrdnun = omp_get_max_threads();
00370      #endif
00371      for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00372           t += ctl.direction * ctl.dt_mod) {
00373          #ifdef ASYNCIO
00374          #pragma omp parallel num_threads(2)
00375          {
00376          #endif
00377
00378              /* Adjust length of final time step... */
00379              if (ctl.direction * (t - ctl.t_stop) > 0)
00380                  t = ctl.t_stop;
00381
00382              /* Set time steps of air parcels... */
00383              module_timesteps(&ctl, atm, met0, dt, t);
00384
00385              /* Get meteo data... */
00386              #ifdef ASYNCIO
00387              #pragma acc wait(5)
00388              #pragma omp barrier
00389              if (omp_get_thread_num() == 0) {
00390
00391                  /* Pointer swap... */
00392                  if (t != ctl.t_start) {
00393                      mets = met0;
00394                      met0 = met0TMP;
00395                      met0TMP = mets;
00396
00397                      mets = met1;
00398                      met1 = met1TMP;
00399                      met1TMP = mets;
00400                  }
00401              #endif
00402              #ifndef ASYNCIO
00403                  if (t != ctl.t_start)
00404                      get_met(&ctl, clim, t, &met0, &met1);
00405              #endif
00406
00407              /* Sort particles... */
00408              if (ctl.sort_dt > 0 && fmod(t, ctl.sort_dt) == 0)
00409                  module_sort(&ctl, met0, atm);

```



```

00410
00411     /* Check initial positions... */
00412     module_position(&ctl, met0, met1, atm, dt);
00413
00414     /* Advection... */
00415     module_advect(&ctl, met0, met1, atm, dt);
00416
00417     /* Turbulent diffusion... */
00418     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00419         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
00420         module_diffusion_turb(&ctl, clim, atm, dt, rs);
00421
00422     /* Mesoscale diffusion... */
00423     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
00424         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00425
00426     /* Convection... */
00427     if (ctl.conv_cape >= 0
00428         && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
00429         module_convection(&ctl, met0, met1, atm, dt, rs);
00430
00431     /* Sedimentation... */
00432     if (ctl.qnt_rp >= 0 && ctl.qnt_rhop >= 0)
00433         module_sedi(&ctl, met0, met1, atm, dt);
00434
00435     /* Isosurface... */
00436     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00437         module_isosurf(&ctl, met0, met1, atm, cache);
00438
00439     /* Check final positions... */
00440     module_position(&ctl, met0, met1, atm, dt);
00441
00442     /* Interpolate meteo data... */
00443     if (ctl.met_dt_out > 0
00444         && (ctl.met_dt_out < ctl.dt_mod
00445             || fmod(t, ctl.met_dt_out) == 0))
00446         module_meteo(&ctl, clim, met0, met1, atm);
00447
00448     /* Decay of particle mass... */
00449     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00450         module_decay(&ctl, clim, atm, dt);
00451
00452     /* OH chemistry... */
00453     if (ctl.clim_oh_filename[0] != '-' && ctl.oh_chem_reaction != 0)
00454         module_oh_chem(&ctl, clim, met0, met1, atm, dt);
00455
00456     /* H2O2 chemistry (for SO2 aqueous phase oxidation)... */
00457     if (ctl.clim_h2o2_filename[0] != '-' && ctl.h2o2_chem_reaction != 0) {
00458         module_chemgrid(&ctl, met0, met1, atm, t);
00459         module_h2o2_chem(&ctl, clim, met0, met1, atm, dt, rs);
00460     }
00461
00462     /* Dry deposition... */
00463     if (ctl.dry_depo_vdep > 0)
00464         module_dry_deposition(&ctl, met0, met1, atm, dt);
00465
00466     /* Wet deposition... */
00467     if ((ctl.wet_depo_ic_a > 0 || ctl.wet_depo_ic_h[0] > 0)
00468         && (ctl.wet_depo_bc_a > 0 || ctl.wet_depo_bc_h[0] > 0))
00469         module_wet_deposition(&ctl, met0, met1, atm, dt);
00470
00471     /* Boundary conditions... */
00472     if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
00473         module_bound_cond(&ctl, met0, met1, atm, dt);
00474
00475     /* Write output... */
00476     write_output(dirname, &ctl, met0, met1, atm, t);
00477     #ifdef ASYNCIO
00478     } else {
00479         omp_set_num_threads(ompTrdnum);
00480         if (ctl.direction * (t - ctl.t_stop + ctl.direction * ctl.dt_mod) <
00481             ctl.dt_mod)
00482             get_met(&ctl, clim, t + (ctl.direction * ctl.dt_mod), &met0TMP,
00483                 &met1TMP);
00484     }
00485     }
00486 #endif
00487 }
00488
00489 #ifdef ASYNCIO
00490     omp_set_num_threads(ompTrdnum);
00491 #endif
00492
00493     /* -----
00494     Finalize model run...
00495     ----- */
00496

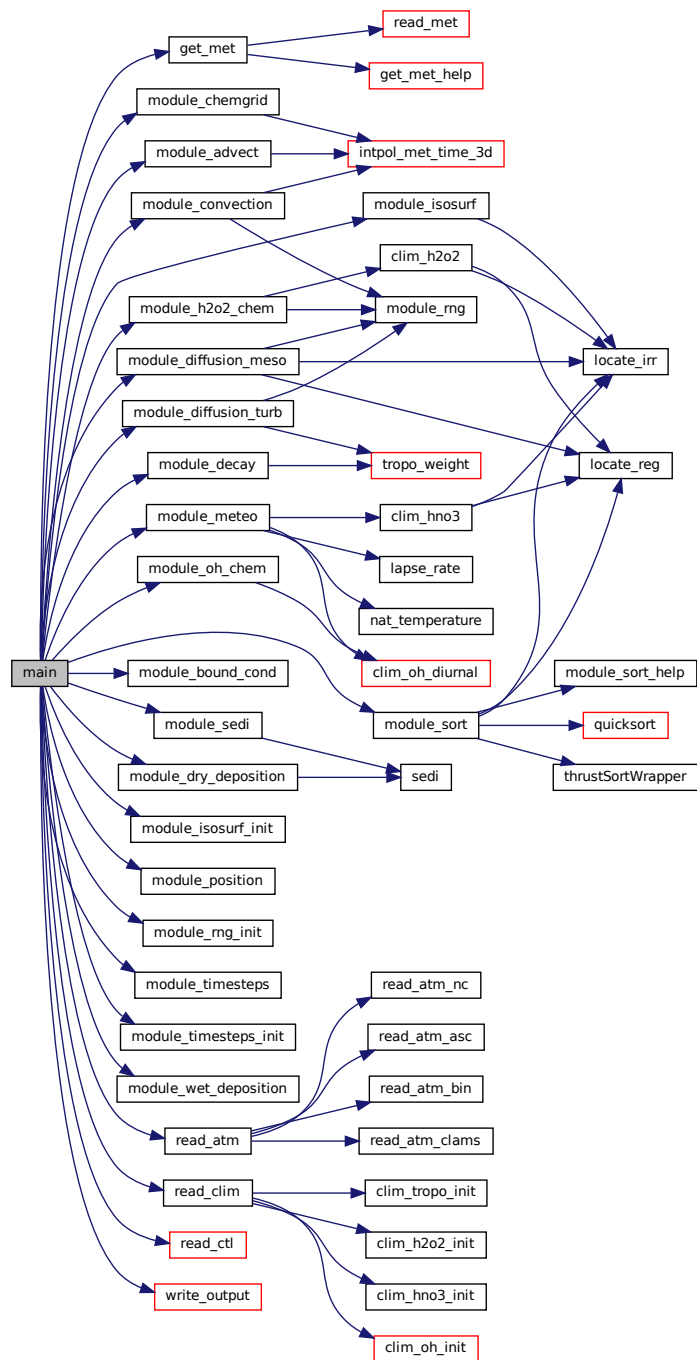
```

```

00497     /* Report problem size... */
00498     LOG(1, "SIZE_NP = %d", atm->np);
00499     LOG(1, "SIZE_MPI_TASKS = %d", size);
00500     LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
00501 #ifndef _OPENACC
00502     LOG(1, "SIZE_ACC_DEVICES = %d", acc_get_num_devices(acc_device_nvidia));
00503 #endif
00504
00505     /* Report memory usage... */
00506     LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
00507     LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
00508     LOG(1, "MEMORY_CLIM = %g MByte", sizeof(clim_t) / 1024. / 1024.);
00509     LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
00510     LOG(1, "MEMORY_DYNAMIC = %g MByte", (3 * NP * sizeof(int)
00511                                         + 4 * NP * sizeof(double)
00512                                         + EX * EY * EP * sizeof(float)) /
00513         1024. / 1024.);
00514     LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * EP * sizeof(float)) /
00515         1024. / 1024.);
00516
00517     /* Delete data region on GPUs... */
00518 #ifndef _OPENACC
00519     SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
00520 #endif
00521 #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs, met0TMP, met1TMP)
00522 #else
00523 #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs)
00524 #endif
00525 #endif
00526
00527     /* Free... */
00528     SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
00529     free(atm);
00530     free(cache);
00531     free(clim);
00532     free(met0);
00533     free(met1);
00534 #ifndef ASYNCIO
00535     free(met0TMP);
00536     free(met1TMP);
00537 #endif
00538     free(dt);
00539     free(rs);
00540
00541     /* Report timers... */
00542     PRINT_TIMERS;
00543 }
00544
00545     /* Finalize MPI... */
00546 #ifndef MPI
00547     MPI_Finalize();
00548 #endif
00549
00550     /* Stop timers... */
00551     STOP_TIMERS;
00552
00553     return EXIT_SUCCESS;
00554 }

```

Here is the call graph for this function:



5.48 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) 2013-2023 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(
00043     ctl_t * ctl,
00044     met_t * met0,
00045     met_t * met1,
00046     atm_t * atm,
00047     double *dt);
00048
00050 void module_bound_cond(
00051     ctl_t * ctl,
00052     met_t * met0,
00053     met_t * met1,
00054     atm_t * atm,
00055     double *dt);
00056
00058 void module_convection(
00059     ctl_t * ctl,
00060     met_t * met0,
00061     met_t * met1,
00062     atm_t * atm,
00063     double *dt,
00064     double *rs);
00065
00067 void module_decay(
00068     ctl_t * ctl,
00069     clim_t * clim,
00070     atm_t * atm,
00071     double *dt);
00072
00074 void module_diffusion_meso(
00075     ctl_t * ctl,
00076     met_t * met0,
00077     met_t * met1,
00078     atm_t * atm,
00079     cache_t * cache,
00080     double *dt,
00081     double *rs);
00082
00084 void module_diffusion_turb(
00085     ctl_t * ctl,
00086     clim_t * clim,
00087     atm_t * atm,
00088     double *dt,
00089     double *rs);
00090
00092 void module_dry_deposition(
00093     ctl_t * ctl,
00094     met_t * met0,
00095     met_t * met1,
00096     atm_t * atm,
00097     double *dt);
00098
00100 void module_isosurf_init(
00101     ctl_t * ctl,
00102     met_t * met0,
00103     met_t * met1,
00104     atm_t * atm,
00105     cache_t * cache);
00106
00108 void module_isosurf(
00109     ctl_t * ctl,

```

```
00110     met_t * met0,
00111     met_t * met1,
00112     atm_t * atm,
00113     cache_t * cache);
00114
00116 void module_meteo(
00117     ctl_t * ctl,
00118     clim_t * clim,
00119     met_t * met0,
00120     met_t * met1,
00121     atm_t * atm);
00122
00124 void module_oh_chem(
00125     ctl_t * ctl,
00126     clim_t * clim,
00127     met_t * met0,
00128     met_t * met1,
00129     atm_t * atm,
00130     double *dt);
00131
00133 void module_h2o2_chem(
00134     ctl_t * ctl,
00135     clim_t * clim,
00136     met_t * met0,
00137     met_t * met1,
00138     atm_t * atm,
00139     double *dt,
00140     double *rs);
00141
00143 void module_chemgrid(
00144     ctl_t * ctl,
00145     met_t * met0,
00146     met_t * met1,
00147     atm_t * atm,
00148     double t);
00149
00151 void module_position(
00152     ctl_t * ctl,
00153     met_t * met0,
00154     met_t * met1,
00155     atm_t * atm,
00156     double *dt);
00157
00159 void module_rng_init(
00160     int ntask);
00161
00163 void module_rng(
00164     double *rs,
00165     size_t n,
00166     int method);
00167
00169 void module_sedi(
00170     ctl_t * ctl,
00171     met_t * met0,
00172     met_t * met1,
00173     atm_t * atm,
00174     double *dt);
00175
00177 void module_sort(
00178     ctl_t * ctl,
00179     met_t * met0,
00180     atm_t * atm);
00181
00183 void module_sort_help(
00184     double *a,
00185     int *p,
00186     int np);
00187
00189 void module_timesteps(
00190     ctl_t * ctl,
00191     atm_t * atm,
00192     met_t * met0,
00193     double *dt,
00194     double t);
00195
00197 void module_timesteps_init(
00198     ctl_t * ctl,
00199     atm_t * atm);
00200
00202 void module_wet_deposition(
00203     ctl_t * ctl,
00204     met_t * met0,
00205     met_t * met1,
00206     atm_t * atm,
00207     double *dt);
00208
00210 void write_output(
```

```

00211     const char *dirname,
00212     ctl_t *ctl,
00213     met_t *met0,
00214     met_t *met1,
00215     atm_t *atm,
00216     double t);
00217
00218 /* -----
00219     Main...
00220     ----- */
00221
00222 int main(
00223     int argc,
00224     char *argv[]) {
00225
00226     ctl_t ctl;
00227
00228     atm_t *atm;
00229
00230     cache_t *cache;
00231
00232     clim_t *clim;
00233
00234     met_t *met0, *met1;
00235
00236 #ifdef ASYNCIO
00237     met_t *met0TMP, *met1TMP, *mets;
00238     ctl_t ctlTMP;
00239 #endif
00240
00241     FILE *dirlist;
00242
00243     char dirname[LEN], filename[2 * LEN];
00244
00245     double *dt, *rs, t;
00246
00247     int ntask = -1, rank = 0, size = 1;
00248
00249     /* Start timers... */
00250     START_TIMERS;
00251
00252     /* Initialize MPI... */
00253 #ifdef MPI
00254     SELECT_TIMER("MPI_INIT", "INIT", NVTX_CPU);
00255     MPI_Init(&argc, &argv);
00256     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00257     MPI_Comm_size(MPI_COMM_WORLD, &size);
00258 #endif
00259
00260     /* Initialize GPUs... */
00261 #ifdef _OPENACC
00262     SELECT_TIMER("ACC_INIT", "INIT", NVTX_GPU);
00263     if (acc_get_num_devices(acc_device_nvidia) <= 0)
00264         ERRMSG("Not running on a GPU device!");
00265     acc_set_device_num(rank % acc_get_num_devices(acc_device_nvidia),
00266                       acc_device_nvidia);
00267     acc_device_t device_type = acc_get_device_type();
00268     acc_init(device_type);
00269 #endif
00270
00271     /* Check arguments... */
00272     if (argc < 4)
00273         ERRMSG("Give parameters: <dirlist> <ctl> <atm_in>");
00274
00275     /* Open directory list... */
00276     if (!(dirlist = fopen(argv[1], "r")))
00277         ERRMSG("Cannot open directory list!");
00278
00279     /* Loop over directories... */
00280     while (fscanf(dirlist, "%4999s", dirname) != EOF) {
00281
00282         /* MPI parallelization... */
00283         if ((++ntask) % size != rank)
00284             continue;
00285
00286         /* -----
00287             Initialize model run...
00288             ----- */
00289
00290         /* Allocate... */
00291         SELECT_TIMER("ALLOC", "MEMORY", NVTX_CPU);
00292         ALLOC(atm, atm_t, 1);
00293         ALLOC(cache, cache_t, 1);
00294         ALLOC(clim, clim_t, 1);
00295         ALLOC(met0, met_t, 1);
00296         ALLOC(met1, met_t, 1);
00297 #ifdef ASYNCIO

```

```

00298     ALLOC(met0TMP, met_t, 1);
00299     ALLOC(met1TMP, met_t, 1);
00300 #endif
00301     ALLOC(dt, double,
00302           NP);
00303     ALLOC(rs, double,
00304           3 * NP + 1);
00305
00306     /* Create data region on GPUs... */
00307 #ifdef _OPENACC
00308     SELECT_TIMER("CREATE_DATA_REGION", "MEMORY", NVTX_GPU);
00309 #ifdef ASYNCIO
00310 #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl,ctlTMP, met0[:1], met1[:1],
00311                             met0TMP[:1], met1TMP[:1], dt[:NP], rs[:3 * NP])
00312 #else
00312 #pragma acc enter data create(atm[:1], cache[:1], clim[:1], ctl, met0[:1], met1[:1], dt[:NP], rs[:3 *
00313                             NP])
00314 #endif
00315
00316     /* Read control parameters... */
00317     sprintf(filename, "%s/%s", dirname, argv[2]);
00318     read_ctl(filename, argc, argv, &ctl);
00319
00320     /* Read climatological data... */
00321     read_clim(&ctl, clim);
00322
00323     /* Read atmospheric data... */
00324     sprintf(filename, "%s/%s", dirname, argv[3]);
00325     if (!read_atm(filename, &ctl, atm))
00326         ERRMSG("Cannot open file!");
00327
00328     /* Initialize timesteps... */
00329     module_timesteps_init(&ctl, atm);
00330
00331     /* Update GPU... */
00332 #ifdef _OPENACC
00333     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00334 #pragma acc update device(atm[:1], clim[:1], ctl)
00335 #endif
00336
00337     /* Initialize random number generator... */
00338     module_rng_init(ntask);
00339
00340     /* Initialize meteo data... */
00341 #ifdef ASYNCIO
00342     ctlTMP = ctl;
00343 #endif
00344     get_met(&ctl, clim, ctl.t_start, &met0, &met1);
00345     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00346         WARN("Violation of CFL criterion! Check DT_MOD!");
00347 #ifdef ASYNCIO
00348     get_met(&ctlTMP, clim, ctlTMP.t_start, &met0TMP, &met1TMP);
00349 #endif
00350
00351     /* Initialize isosurface... */
00352     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00353         module_isosurf_init(&ctl, met0, met1, atm, cache);
00354
00355     /* Update GPU... */
00356 #ifdef _OPENACC
00357     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
00358 #pragma acc update device(cache[:1])
00359 #endif
00360
00361     /* -----
00362     Loop over timesteps...
00363     ----- */
00364
00365     /* Loop over timesteps... */
00366 #ifdef ASYNCIO
00367     omp_set_nested(1);
00368     // omp_set_dynamic(0);
00369     int ompTrdnum = omp_get_max_threads();
00370 #endif
00371     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00372          t += ctl.direction * ctl.dt_mod) {
00373 #ifdef ASYNCIO
00374 #pragma omp parallel num_threads(2)
00375 {
00376 #endif
00377
00378     /* Adjust length of final time step... */
00379     if (ctl.direction * (t - ctl.t_stop) > 0)
00380         t = ctl.t_stop;
00381
00382     /* Set time steps of air parcels... */

```

```

00383     module_timesteps(&ctl, atm, met0, dt, t);
00384
00385     /* Get meteo data... */
00386 #ifdef ASYNCIO
00387 #pragma acc wait(5)
00388 #pragma omp barrier
00389     if (omp_get_thread_num() == 0) {
00390
00391         /* Pointer swap... */
00392         if (t != ctl.t_start) {
00393             mets = met0;
00394             met0 = met0TMP;
00395             met0TMP = mets;
00396
00397             mets = met1;
00398             met1 = met1TMP;
00399             met1TMP = mets;
00400         }
00401 #endif
00402 #ifndef ASYNCIO
00403     if (t != ctl.t_start)
00404         get_met(&ctl, clim, t, &met0, &met1);
00405 #endif
00406
00407     /* Sort particles... */
00408     if (ctl.sort_dt > 0 && fmod(t, ctl.sort_dt) == 0)
00409         module_sort(&ctl, met0, atm);
00410
00411     /* Check initial positions... */
00412     module_position(&ctl, met0, met1, atm, dt);
00413
00414     /* Advection... */
00415     module_advect(&ctl, met0, met1, atm, dt);
00416
00417     /* Turbulent diffusion... */
00418     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00419         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0)
00420         module_diffusion_turb(&ctl, clim, atm, dt, rs);
00421
00422     /* Mesoscale diffusion... */
00423     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0)
00424         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00425
00426     /* Convection... */
00427     if (ctl.conv_cape >= 0
00428         && (ctl.conv_dt <= 0 || fmod(t, ctl.conv_dt) == 0))
00429         module_convection(&ctl, met0, met1, atm, dt, rs);
00430
00431     /* Sedimentation... */
00432     if (ctl.qnt_rp >= 0 && ctl.qnt_rhop >= 0)
00433         module_sedi(&ctl, met0, met1, atm, dt);
00434
00435     /* Isosurface... */
00436     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00437         module_isosurf(&ctl, met0, met1, atm, cache);
00438
00439     /* Check final positions... */
00440     module_position(&ctl, met0, met1, atm, dt);
00441
00442     /* Interpolate meteo data... */
00443     if (ctl.met_dt_out > 0
00444         && (ctl.met_dt_out < ctl.dt_mod
00445             || fmod(t, ctl.met_dt_out) == 0))
00446         module_meteo(&ctl, clim, met0, met1, atm);
00447
00448     /* Decay of particle mass... */
00449     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00450         module_decay(&ctl, clim, atm, dt);
00451
00452     /* OH chemistry... */
00453     if (ctl.clim_oh_filename[0] != '-' && ctl.oh_chem_reaction != 0)
00454         module_oh_chem(&ctl, clim, met0, met1, atm, dt);
00455
00456     /* H2O2 chemistry (for SO2 aqueous phase oxidation)... */
00457     if (ctl.clim_h2o2_filename[0] != '-' && ctl.h2o2_chem_reaction != 0) {
00458         module_chemgrid(&ctl, met0, met1, atm, t);
00459         module_h2o2_chem(&ctl, clim, met0, met1, atm, dt, rs);
00460     }
00461
00462     /* Dry deposition... */
00463     if (ctl.dry_depo_vdep > 0)
00464         module_dry_deposition(&ctl, met0, met1, atm, dt);
00465
00466     /* Wet deposition... */
00467     if ((ctl.wet_depo_ic_a > 0 || ctl.wet_depo_ic_h[0] > 0)
00468         && (ctl.wet_depo_bc_a > 0 || ctl.wet_depo_bc_h[0] > 0))
00469         module_wet_deposition(&ctl, met0, met1, atm, dt);

```



```

00470
00471     /* Boundary conditions... */
00472     if (ctl.bound_mass >= 0 || ctl.bound_vmr >= 0)
00473         module_bound_cond(&ctl, met0, met1, atm, dt);
00474
00475     /* Write output... */
00476     write_output(dirname, &ctl, met0, met1, atm, t);
00477 #ifdef ASYNCIO
00478     } else {
00479         omp_set_num_threads(ompTrdnm);
00480         if (ctl.direction * (t - ctl.t_stop + ctl.direction * ctl.dt_mod) <
00481             ctl.dt_mod)
00482             get_met(&ctl, clim, t + (ctl.direction * ctl.dt_mod), &met0TMP,
00483                 &met1TMP);
00484     }
00485 }
00486 #endif
00487 }
00488
00489 #ifdef ASYNCIO
00490     omp_set_num_threads(ompTrdnm);
00491 #endif
00492
00493     /* -----
00494     Finalize model run...
00495     ----- */
00496
00497     /* Report problem size... */
00498     LOG(1, "SIZE_NP = %d", atm->np);
00499     LOG(1, "SIZE_MPI_TASKS = %d", size);
00500     LOG(1, "SIZE_OMP_THREADS = %d", omp_get_max_threads());
00501 #ifdef _OPENACC
00502     LOG(1, "SIZE_ACC_DEVICES = %d", acc_get_num_devices(acc_device_nvidia));
00503 #endif
00504
00505     /* Report memory usage... */
00506     LOG(1, "MEMORY_ATM = %g MByte", sizeof(atm_t) / 1024. / 1024.);
00507     LOG(1, "MEMORY_CACHE = %g MByte", sizeof(cache_t) / 1024. / 1024.);
00508     LOG(1, "MEMORY_CLIM = %g MByte", sizeof(clim_t) / 1024. / 1024.);
00509     LOG(1, "MEMORY_METEO = %g MByte", 2 * sizeof(met_t) / 1024. / 1024.);
00510     LOG(1, "MEMORY_DYNAMIC = %g MByte", (3 * NP * sizeof(int)
00511         + 4 * NP * sizeof(double)
00512         + EX * EY * EP * sizeof(float)) /
00513         1024. / 1024.);
00514     LOG(1, "MEMORY_STATIC = %g MByte", (EX * EY * EP * sizeof(float)) /
00515         1024. / 1024.);
00516
00517     /* Delete data region on GPUs... */
00518 #ifdef _OPENACC
00519     SELECT_TIMER("DELETE_DATA_REGION", "MEMORY", NVTX_GPU);
00520 #ifdef ASYNCIO
00521 #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs, met0TMP, met1TMP)
00522 #else
00523 #pragma acc exit data delete (ctl, atm, cache, clim, met0, met1, dt, rs)
00524 #endif
00525 #endif
00526
00527     /* Free... */
00528     SELECT_TIMER("FREE", "MEMORY", NVTX_CPU);
00529     free(atm);
00530     free(cache);
00531     free(clim);
00532     free(met0);
00533     free(met1);
00534 #ifdef ASYNCIO
00535     free(met0TMP);
00536     free(met1TMP);
00537 #endif
00538     free(dt);
00539     free(rs);
00540
00541     /* Report timers... */
00542     PRINT_TIMERS;
00543 }
00544
00545     /* Finalize MPI... */
00546 #ifdef MPI
00547     MPI_Finalize();
00548 #endif
00549
00550     /* Stop timers... */
00551     STOP_TIMERS;
00552
00553     return EXIT_SUCCESS;
00554 }
00555
00556 /*****

```

```

00557
00558 void module_advect(
00559     ctl_t * ctl,
00560     met_t * met0,
00561     met_t * met1,
00562     atm_t * atm,
00563     double *dt) {
00564
00565     /* Set timer... */
00566     SELECT_TIMER("MODULE_ADVECTION", "PHYSICS", NVTX_GPU);
00567
00568     const int np = atm->np;
00569 #ifdef _OPENACC
00570 #pragma acc data present(ctl,met0,met1,atm,dt)
00571 #pragma acc parallel loop independent gang vector
00572 #else
00573 #pragma omp parallel for default(shared)
00574 #endif
00575     for (int ip = 0; ip < np; ip++)
00576         if (dt[ip] != 0) {
00577
00578             /* Init... */
00579             double dts, u[4], um = 0, v[4], vm = 0, w[4], wm = 0, x[3];
00580
00581             /* Loop over integration nodes... */
00582             for (int i = 0; i < ctl->advect; i++) {
00583
00584                 /* Set position... */
00585                 if (i == 0) {
00586                     dts = 0.0;
00587                     x[0] = atm->lon[ip];
00588                     x[1] = atm->lat[ip];
00589                     x[2] = atm->p[ip];
00590                 } else {
00591                     dts = (i == 3 ? 1.0 : 0.5) * dt[ip];
00592                     x[0] = atm->lon[ip] + DX2DEG(dts * u[i - 1] / 1000., atm->lat[ip]);
00593                     x[1] = atm->lat[ip] + DY2DEG(dts * v[i - 1] / 1000.);
00594                     x[2] = atm->p[ip] + dts * w[i - 1];
00595                 }
00596                 double tm = atm->time[ip] + dts;
00597
00598                 /* Interpolate meteo data... */
00599 #ifdef UVW
00600                 intpol_met_time_uvw(met0, met1, tm, x[2], x[0], x[1],
00601                                     &u[i], &v[i], &w[i]);
00602 #else
00603                 INTPOL_INIT;
00604                 intpol_met_time_3d(met0, met0->u, met1, met1->u, tm,
00605                                   x[2], x[0], x[1], &u[i], ci, cw, 1);
00606                 intpol_met_time_3d(met0, met0->v, met1, met1->v, tm,
00607                                   x[2], x[0], x[1], &v[i], ci, cw, 0);
00608                 intpol_met_time_3d(met0, met0->w, met1, met1->w, tm,
00609                                   x[2], x[0], x[1], &w[i], ci, cw, 0);
00610 #endif
00611
00612                 /* Get mean wind... */
00613                 double k = 1.0;
00614                 if (ctl->advect == 2)
00615                     k = (i == 0 ? 0.0 : 1.0);
00616                 else if (ctl->advect == 4)
00617                     k = (i == 0 || i == 3 ? 1.0 / 6.0 : 2.0 / 6.0);
00618                 um += k * u[i];
00619                 vm += k * v[i];
00620                 wm += k * w[i];
00621             }
00622
00623             /* Set new position... */
00624             atm->time[ip] += dt[ip];
00625             atm->lon[ip] += DX2DEG(dt[ip] * um / 1000.,
00626                                   (ctl->advect == 2 ? x[1] : atm->lat[ip]));
00627             atm->lat[ip] += DY2DEG(dt[ip] * vm / 1000.);
00628             atm->p[ip] += dt[ip] * wm;
00629         }
00630     }
00631
00632     /*****
00633
00634 void module_bound_cond(
00635     ctl_t * ctl,
00636     met_t * met0,
00637     met_t * met1,
00638     atm_t * atm,
00639     double *dt) {
00640
00641     /* Set timer... */
00642     SELECT_TIMER("MODULE_BOUNDCOND", "PHYSICS", NVTX_GPU);
00643

```

```

00644  /* Check quantity flags... */
00645  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00646      ERRMSG("Module needs quantity mass or volume mixing ratio!");
00647
00648  const int np = atm->np;
00649  #ifdef _OPENACC
00650  #pragma acc data present(ctl, met0, met1, atm, dt)
00651  #pragma acc parallel loop independent gang vector
00652  #else
00653  #pragma omp parallel for default(shared)
00654  #endif
00655  for (int ip = 0; ip < np; ip++)
00656      if (dt[ip] != 0) {
00657
00658          /* Check latitude and pressure range... */
00659          if (atm->lat[ip] < ctl->bound_lat0 || atm->lat[ip] > ctl->bound_lat1
00660              || atm->p[ip] > ctl->bound_p0 || atm->p[ip] < ctl->bound_p1)
00661              continue;
00662
00663          /* Check surface layer... */
00664          if (ctl->bound_dps > 0 || ctl->bound_dzs > 0
00665              || ctl->bound_zetas > 0 || ctl->bound_pbl) {
00666
00667              /* Get surface pressure... */
00668              double ps;
00669              INTPOL_INIT;
00670              INTPOL_2D(ps, 1);
00671
00672              /* Check pressure... */
00673              if (ctl->bound_dps > 0 && atm->p[ip] < ps - ctl->bound_dps)
00674                  continue;
00675
00676              /* Check height... */
00677              if (ctl->bound_dzs > 0 && Z(atm->p[ip]) > Z(ps) + ctl->bound_dzs)
00678                  continue;
00679
00680              /* Check zeta range... */
00681              if (ctl->bound_zetas > 0) {
00682                  double t;
00683                  INTPOL_3D(t, 1);
00684                  if (ZETA(ps, atm->p[ip], t) > ctl->bound_zetas)
00685                      continue;
00686              }
00687
00688              /* Check planetary boundary layer... */
00689              if (ctl->bound_pbl) {
00690                  double pbl;
00691                  INTPOL_2D(pbl, 0);
00692                  if (atm->p[ip] < pbl)
00693                      continue;
00694              }
00695          }
00696
00697          /* Set mass and volume mixing ratio... */
00698          if (ctl->qnt_m >= 0 && ctl->bound_mass >= 0)
00699              atm->q[ctl->qnt_m][ip] =
00700                  ctl->bound_mass + ctl->bound_mass_trend * atm->time[ip];
00701          if (ctl->qnt_vmr >= 0 && ctl->bound_vmr >= 0)
00702              atm->q[ctl->qnt_vmr][ip] =
00703                  ctl->bound_vmr + ctl->bound_vmr_trend * atm->time[ip];
00704      }
00705 }
00706
00707 /*****
00708
00709 void module_convection(
00710     ctl_t * ctl,
00711     met_t * met0,
00712     met_t * met1,
00713     atm_t * atm,
00714     double *dt,
00715     double *rs) {
00716
00717     /* Set timer... */
00718     SELECT_TIMER("MODULE_CONVECTION", "PHYSICS", NVTX_GPU);
00719
00720     /* Create random numbers... */
00721     module_rng(rs, (size_t) atm->np, 0);
00722
00723     const int np = atm->np;
00724     #ifdef _OPENACC
00725     #pragma acc data present(ctl, met0, met1, atm, dt, rs)
00726     #pragma acc parallel loop independent gang vector
00727     #else
00728     #pragma omp parallel for default(shared)
00729     #endif
00730     for (int ip = 0; ip < np; ip++)

```

```

00731     if (dt[ip] != 0) {
00732
00733         double cape, cin, pel, ps;
00734
00735         /* Interpolate CAPE... */
00736         INTPOL_INIT;
00737         INTPOL_2D(cape, 1);
00738
00739         /* Check threshold... */
00740         if (isfinite(cape) && cape >= ctl->conv_cape) {
00741
00742             /* Check CIN... */
00743             if (ctl->conv_cin > 0) {
00744                 INTPOL_2D(cin, 0);
00745                 if (isfinite(cin) && cin >= ctl->conv_cin)
00746                     continue;
00747             }
00748
00749             /* Interpolate equilibrium level... */
00750             INTPOL_2D(pel, 0);
00751
00752             /* Check whether particle is above cloud top... */
00753             if (!isfinite(pel) || atm->p[ip] < pel)
00754                 continue;
00755
00756             /* Set pressure range for mixing... */
00757             double pbot = atm->p[ip];
00758             double ptop = atm->p[ip];
00759             if (ctl->conv_mix_bot == 1) {
00760                 INTPOL_2D(ps, 0);
00761                 pbot = ps;
00762             }
00763             if (ctl->conv_mix_top == 1)
00764                 ptop = pel;
00765
00766             /* Vertical mixing based on pressure... */
00767             if (ctl->conv_mix == 0)
00768                 atm->p[ip] = pbot + (ptop - pbot) * rs[ip];
00769
00770             /* Vertical mixing based on density... */
00771             else if (ctl->conv_mix == 1) {
00772
00773                 /* Get density range... */
00774                 double tbot, ttop;
00775                 intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00776                                 pbot, atm->lon[ip], atm->lat[ip], &tbot,
00777                                 ci, cw, 1);
00778                 intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->time[ip],
00779                                 ptop, atm->lon[ip], atm->lat[ip], &ttop,
00780                                 ci, cw, 1);
00781                 double rhobot = pbot / tbot;
00782                 double rhotop = ptop / ttop;
00783
00784                 /* Get new density... */
00785                 double lrho = log(rhobot + (rhotop - rhobot) * rs[ip]);
00786
00787                 /* Find pressure... */
00788                 double lrhobot = log(rhobot);
00789                 double lrhotop = log(rhotop);
00790                 double lpbot = log(pbot);
00791                 double lptop = log(ptop);
00792                 atm->p[ip] = exp(LIN(lrhobot, lpbot, lrhotop, lptop, lrho));
00793             }
00794         }
00795     }
00796 }
00797
00798 /*****
00799
00800 void module_decay(
00801     ctl_t * ctl,
00802     clim_t * clim,
00803     atm_t * atm,
00804     double *dt) {
00805
00806     /* Set timer... */
00807     SELECT_TIMER("MODULE_DECAY", "PHYSICS", NVTX_GPU);
00808
00809     /* Check quantity flags... */
00810     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
00811         ERRMSG("Module needs quantity mass or volume mixing ratio!");
00812
00813     const int np = atm->np;
00814 #ifdef _OPENACC
00815     #pragma acc data present(ctl,clim,atm,dt)
00816     #pragma acc parallel loop independent gang vector
00817 #else

```

```

00818 #pragma omp parallel for default(shared)
00819 #endif
00820 for (int ip = 0; ip < np; ip++)
00821     if (dt[ip] != 0) {
00822
00823         /* Get weighting factor... */
00824         double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00825
00826         /* Set lifetime... */
00827         double tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00828
00829         /* Calculate exponential decay... */
00830         double aux = exp(-dt[ip] / tdec);
00831         if (ctl->qnt_m >= 0) {
00832             if (ctl->qnt_mloss_decay >= 0)
00833                 atm->q[ctl->qnt_mloss_decay][ip]
00834                 += atm->q[ctl->qnt_m][ip] * (1 - aux);
00835             atm->q[ctl->qnt_m][ip] *= aux;
00836         }
00837         if (ctl->qnt_vmr >= 0)
00838             atm->q[ctl->qnt_vmr][ip] *= aux;
00839     }
00840 }
00841
00842 /*****
00843
00844 void module_diffusion_meso(
00845     ctl_t * ctl,
00846     met_t * met0,
00847     met_t * met1,
00848     atm_t * atm,
00849     cache_t * cache,
00850     double *dt,
00851     double *rs) {
00852
00853     /* Set timer... */
00854     SELECT_TIMER("MODULE_TURBMESO", "PHYSICS", NVTX_GPU);
00855
00856     /* Create random numbers... */
00857     module_rng(rs, 3 * (size_t) atm->np, 1);
00858
00859     const int np = atm->np;
00860 #ifdef _OPENACC
00861 #pragma acc data present(ctl, met0, met1, atm, cache, dt, rs)
00862 #pragma acc parallel loop independent gang vector
00863 #else
00864 #pragma omp parallel for default(shared)
00865 #endif
00866     for (int ip = 0; ip < np; ip++)
00867         if (dt[ip] != 0) {
00868
00869             /* Get indices... */
00870             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00871             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00872             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00873
00874             /* Get standard deviations of local wind data... */
00875             float umean = 0, usig = 0, vmean = 0, vsig = 0, wmean = 0, wsig = 0;
00876             for (int i = 0; i < 2; i++)
00877                 for (int j = 0; j < 2; j++)
00878                     for (int k = 0; k < 2; k++) {
00879 #ifdef UVW
00880                         umean += met0->uvw[ix + i][iy + j][iz + k][0];
00881                         usig += SQR(met0->uvw[ix + i][iy + j][iz + k][0]);
00882                         vmean += met0->uvw[ix + i][iy + j][iz + k][1];
00883                         vsig += SQR(met0->uvw[ix + i][iy + j][iz + k][1]);
00884                         wmean += met0->uvw[ix + i][iy + j][iz + k][2];
00885                         wsig += SQR(met0->uvw[ix + i][iy + j][iz + k][2]);
00886
00887                         umean += met1->uvw[ix + i][iy + j][iz + k][0];
00888                         usig += SQR(met1->uvw[ix + i][iy + j][iz + k][0]);
00889                         vmean += met1->uvw[ix + i][iy + j][iz + k][1];
00890                         vsig += SQR(met1->uvw[ix + i][iy + j][iz + k][1]);
00891                         wmean += met1->uvw[ix + i][iy + j][iz + k][2];
00892                         wsig += SQR(met1->uvw[ix + i][iy + j][iz + k][2]);
00893 #else
00894                         umean += met0->u[ix + i][iy + j][iz + k];
00895                         usig += SQR(met0->u[ix + i][iy + j][iz + k]);
00896                         vmean += met0->v[ix + i][iy + j][iz + k];
00897                         vsig += SQR(met0->v[ix + i][iy + j][iz + k]);
00898                         wmean += met0->w[ix + i][iy + j][iz + k];
00899                         wsig += SQR(met0->w[ix + i][iy + j][iz + k]);
00900
00901                         umean += met1->u[ix + i][iy + j][iz + k];
00902                         usig += SQR(met1->u[ix + i][iy + j][iz + k]);
00903                         vmean += met1->v[ix + i][iy + j][iz + k];
00904                         vsig += SQR(met1->v[ix + i][iy + j][iz + k]);

```

```

00905         wmean += met1->w[ix + i][iy + j][iz + k];
00906         wsig += SQR(met1->w[ix + i][iy + j][iz + k]);
00907 #endif
00908     }
00909     usig = usig / 16.f - SQR(umean / 16.f);
00910     usig = (usig > 0 ? sqrtf(usig) : 0);
00911     vsig = vsig / 16.f - SQR(vmean / 16.f);
00912     vsig = (vsig > 0 ? sqrtf(vsig) : 0);
00913     wsig = wsig / 16.f - SQR(wmean / 16.f);
00914     wsig = (wsig > 0 ? sqrtf(wsig) : 0);
00915
00916     /* Set temporal correlations for mesoscale fluctuations... */
00917     double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00918     double r2 = sqrt(1 - r * r);
00919
00920     /* Calculate horizontal mesoscale wind fluctuations... */
00921     if (ctl->turb_mesox > 0) {
00922         cache->uwp[ip][0] =
00923             (float) (r * cache->uwp[ip][0] +
00924                 r2 * rs[3 * ip] * ctl->turb_mesox * usig);
00925         atm->lon[ip] +=
00926             DX2DEG(cache->uwp[ip][0] * dt[ip] / 1000., atm->lat[ip]);
00927
00928         cache->uwp[ip][1] =
00929             (float) (r * cache->uwp[ip][1] +
00930                 r2 * rs[3 * ip + 1] * ctl->turb_mesox * vsig);
00931         atm->lat[ip] += DY2DEG(cache->uwp[ip][1] * dt[ip] / 1000.);
00932     }
00933
00934     /* Calculate vertical mesoscale wind fluctuations... */
00935     if (ctl->turb_mesoz > 0) {
00936         cache->uwp[ip][2] =
00937             (float) (r * cache->uwp[ip][2] +
00938                 r2 * rs[3 * ip + 2] * ctl->turb_mesoz * wsig);
00939         atm->p[ip] += cache->uwp[ip][2] * dt[ip];
00940     }
00941 }
00942 }
00943
00944 /*****
00945
00946 void module_diffusion_turb(
00947     ctl_t * ctl,
00948     clim_t * clim,
00949     atm_t * atm,
00950     double *dt,
00951     double *rs) {
00952
00953     /* Set timer... */
00954     SELECT_TIMER("MODULE_TURBDIFF", "PHYSICS", NVTX_GPU);
00955
00956     /* Create random numbers... */
00957     module_rng(rs, 3 * (size_t) atm->np, 1);
00958
00959     const int np = atm->np;
00960 #ifdef _OPENACC
00961 #pragma acc data present(ctl, clim, atm, dt, rs)
00962 #pragma acc parallel loop independent gang vector
00963 #else
00964 #pragma omp parallel for default(shared)
00965 #endif
00966     for (int ip = 0; ip < np; ip++)
00967         if (dt[ip] != 0) {
00968
00969             /* Get weighting factor... */
00970             double w = tropo_weight(clim, atm->time[ip], atm->lat[ip], atm->p[ip]);
00971
00972             /* Set diffusivity... */
00973             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00974             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00975
00976             /* Horizontal turbulent diffusion... */
00977             if (dx > 0) {
00978                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00979                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00980                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00981             }
00982
00983             /* Vertical turbulent diffusion... */
00984             if (dz > 0) {
00985                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00986                 atm->p[ip] += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00987             }
00988         }
00989 }
00990
00991 /*****

```

```

00992
00993 void module_dry_deposition(
00994     ctl_t * ctl,
00995     met_t * met0,
00996     met_t * met1,
00997     atm_t * atm,
00998     double *dt) {
00999
01000     /* Set timer... */
01001     SELECT_TIMER("MODULE_DRYDEPO", "PHYSICS", NVTX_GPU);
01002
01003     /* Check quantity flags... */
01004     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01005         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01006
01007     const int np = atm->np;
01008     #ifdef _OPENACC
01009     #pragma acc data present(ctl, met0, met1, atm, dt)
01010     #pragma acc parallel loop independent gang vector
01011     #else
01012     #pragma omp parallel for default(shared)
01013     #endif
01014     for (int ip = 0; ip < np; ip++)
01015         if (dt[ip] != 0) {
01016
01017             double ps, t, v_dep;
01018
01019             /* Get surface pressure... */
01020             INTPOL_INIT;
01021             INTPOL_2D(ps, 1);
01022
01023             /* Check whether particle is above the surface layer... */
01024             if (atm->p[ip] < ps - ctl->dry_depo_dp)
01025                 continue;
01026
01027             /* Set depth of surface layer... */
01028             double dz = 1000. * (Z(ps - ctl->dry_depo_dp) - Z(ps));
01029
01030             /* Calculate sedimentation velocity for particles... */
01031             if (ctl->qnt_rp > 0 && ctl->qnt_rhop > 0) {
01032
01033                 /* Get temperature... */
01034                 INTPOL_3D(t, 1);
01035
01036                 /* Set deposition velocity... */
01037                 v_dep = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01038                             atm->q[ctl->qnt_rhop][ip]);
01039             }
01040
01041             /* Use explicit sedimentation velocity for gases... */
01042             else
01043                 v_dep = ctl->dry_depo_vdep;
01044
01045             /* Calculate loss of mass based on deposition velocity... */
01046             double aux = exp(-dt[ip] * v_dep / dz);
01047             if (ctl->qnt_m >= 0) {
01048                 if (ctl->qnt_mloss_dry >= 0)
01049                     atm->q[ctl->qnt_mloss_dry][ip]
01050                         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01051                 atm->q[ctl->qnt_m][ip] *= aux;
01052             }
01053             if (ctl->qnt_vmr >= 0)
01054                 atm->q[ctl->qnt_vmr][ip] *= aux;
01055         }
01056 }
01057
01058 /*****
01059
01060 void module_isosurf_init(
01061     ctl_t * ctl,
01062     met_t * met0,
01063     met_t * met1,
01064     atm_t * atm,
01065     cache_t * cache) {
01066
01067     FILE *in;
01068
01069     char line[LEN];
01070
01071     double t;
01072
01073     /* Set timer... */
01074     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01075
01076     /* Init... */
01077     INTPOL_INIT;
01078

```

```

01079  /* Save pressure... */
01080  if (ctl->isosurf == 1)
01081      for (int ip = 0; ip < atm->np; ip++)
01082          cache->iso_var[ip] = atm->p[ip];
01083
01084  /* Save density... */
01085  else if (ctl->isosurf == 2)
01086      for (int ip = 0; ip < atm->np; ip++) {
01087          INTPOL_3D(t, 1);
01088          cache->iso_var[ip] = atm->p[ip] / t;
01089      }
01090
01091  /* Save potential temperature... */
01092  else if (ctl->isosurf == 3)
01093      for (int ip = 0; ip < atm->np; ip++) {
01094          INTPOL_3D(t, 1);
01095          cache->iso_var[ip] = THETA(atm->p[ip], t);
01096      }
01097
01098  /* Read balloon pressure data... */
01099  else if (ctl->isosurf == 4) {
01100
01101      /* Write info... */
01102      LOG(1, "Read balloon pressure data: %s", ctl->balloon);
01103
01104      /* Open file... */
01105      if (!(in = fopen(ctl->balloon, "r")))
01106          ERRMSG("Cannot open file!");
01107
01108      /* Read pressure time series... */
01109      while (fgets(line, LEN, in))
01110          if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
01111                  &(cache->iso_ps[cache->iso_n])) == 2)
01112              if ((++cache->iso_n) > NP)
01113                  ERRMSG("Too many data points!");
01114
01115      /* Check number of points... */
01116      if (cache->iso_n < 1)
01117          ERRMSG("Could not read any data!");
01118
01119      /* Close file... */
01120      fclose(in);
01121  }
01122 }
01123
01124 /*****
01125
01126 void module_isosurf(
01127     ctl_t * ctl,
01128     met_t * met0,
01129     met_t * met1,
01130     atm_t * atm,
01131     cache_t * cache) {
01132
01133     /* Set timer... */
01134     SELECT_TIMER("MODULE_ISOSURF", "PHYSICS", NVTX_GPU);
01135
01136     const int np = atm->np;
01137     #ifdef _OPENACC
01138     #pragma acc data present(ctl, met0, met1, atm, cache)
01139     #pragma acc parallel loop independent gang vector
01140     #else
01141     #pragma omp parallel for default(shared)
01142     #endif
01143     for (int ip = 0; ip < np; ip++) {
01144
01145         double t;
01146
01147         /* Init... */
01148         INTPOL_INIT;
01149
01150         /* Restore pressure... */
01151         if (ctl->isosurf == 1)
01152             atm->p[ip] = cache->iso_var[ip];
01153
01154         /* Restore density... */
01155         else if (ctl->isosurf == 2) {
01156             INTPOL_3D(t, 1);
01157             atm->p[ip] = cache->iso_var[ip] * t;
01158         }
01159
01160         /* Restore potential temperature... */
01161         else if (ctl->isosurf == 3) {
01162             INTPOL_3D(t, 1);
01163             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
01164         }
01165     }

```



```

01166     /* Interpolate pressure... */
01167     else if (ctl->isosurf == 4) {
01168         if (atm->time[ip] <= cache->iso_ts[0])
01169             atm->p[ip] = cache->iso_ps[0];
01170         else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
01171             atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
01172         else {
01173             int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->time[ip]);
01174             atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
01175                             cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
01176                             atm->time[ip]);
01177         }
01178     }
01179 }
01180 }
01181
01182 /*****
01183
01184 void module_meteo(
01185     ctl_t * ctl,
01186     clim_t * clim,
01187     met_t * met0,
01188     met_t * met1,
01189     atm_t * atm) {
01190
01191     /* Set timer... */
01192     SELECT_TIMER("MODULE_METEO", "PHYSICS", NVTX_GPU);
01193
01194     /* Check quantity flags... */
01195     if (ctl->qnt_tsts >= 0)
01196         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01197             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01198
01199     const int np = atm->np;
01200 #ifdef _OPENACC
01201 #pragma acc data present(ctl, clim, met0, met1, atm)
01202 #pragma acc parallel loop independent gang vector
01203 #else
01204 #pragma omp parallel for default(shared)
01205 #endif
01206     for (int ip = 0; ip < np; ip++) {
01207
01208         double ps, ts, zs, us, vs, pbl, pt, pct, pcb, cl, plcl, plfc, pel, cape,
01209             cin, pv, t, tt, u, v, w, h2o, h2ot, o3, lwc, iwc, z, zt;
01210
01211         /* Interpolate meteo data... */
01212         INTPOL_INIT;
01213         INTPOL_TIME_ALL(atm->time[ip], atm->p[ip], atm->lon[ip], atm->lat[ip]);
01214
01215         /* Set quantities... */
01216         SET_ATM(qnt_ps, ps);
01217         SET_ATM(qnt_ts, ts);
01218         SET_ATM(qnt_zs, zs);
01219         SET_ATM(qnt_us, us);
01220         SET_ATM(qnt_vs, vs);
01221         SET_ATM(qnt_pbl, pbl);
01222         SET_ATM(qnt_pt, pt);
01223         SET_ATM(qnt_tt, tt);
01224         SET_ATM(qnt_zt, zt);
01225         SET_ATM(qnt_h2ot, h2ot);
01226         SET_ATM(qnt_z, z);
01227         SET_ATM(qnt_p, atm->p[ip]);
01228         SET_ATM(qnt_t, t);
01229         SET_ATM(qnt_rho, RHO(atm->p[ip], t));
01230         SET_ATM(qnt_u, u);
01231         SET_ATM(qnt_v, v);
01232         SET_ATM(qnt_w, w);
01233         SET_ATM(qnt_h2o, h2o);
01234         SET_ATM(qnt_o3, o3);
01235         SET_ATM(qnt_lwc, lwc);
01236         SET_ATM(qnt_iwc, iwc);
01237         SET_ATM(qnt_pct, pct);
01238         SET_ATM(qnt_pcb, pcb);
01239         SET_ATM(qnt_cl, cl);
01240         SET_ATM(qnt_plcl, plcl);
01241         SET_ATM(qnt_plfc, plfc);
01242         SET_ATM(qnt_pel, pel);
01243         SET_ATM(qnt_cape, cape);
01244         SET_ATM(qnt_cin, cin);
01245         SET_ATM(qnt_hno3,
01246             clim_hno3(clim, atm->time[ip], atm->lat[ip], atm->p[ip]));
01247         SET_ATM(qnt_oh,
01248             clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01249                             atm->lon[ip], atm->lat[ip]));
01250         SET_ATM(qnt_vh, sqrt(u * u + v * v));
01251         SET_ATM(qnt_vz, -1e3 * H0 / atm->p[ip] * w);
01252         SET_ATM(qnt_psat, PSAT(t));

```

```

01253     SET_ATM(qnt_psice, PSICE(t));
01254     SET_ATM(qnt_pw, PW(atm->p[ip], h2o));
01255     SET_ATM(qnt_sh, SH(h2o));
01256     SET_ATM(qnt_rh, RH(atm->p[ip], t, h2o));
01257     SET_ATM(qnt_rhice, RHICE(atm->p[ip], t, h2o));
01258     SET_ATM(qnt_theta, THETA(atm->p[ip], t));
01259     SET_ATM(qnt_zeta, ZETA(ps, atm->p[ip], t));
01260     SET_ATM(qnt_tvirt, TVIRT(t, h2o));
01261     SET_ATM(qnt_lapse, lapse_rate(t, h2o));
01262     SET_ATM(qnt_pv, pv);
01263     SET_ATM(qnt_tdew, TDEW(atm->p[ip], h2o));
01264     SET_ATM(qnt_tice, TICE(atm->p[ip], h2o));
01265     SET_ATM(qnt_tnat,
01266             nat_temperature(atm->p[ip], h2o,
01267                             clim_hno3(clim, atm->time[ip], atm->lat[ip],
01268                                         atm->p[ip])));
01269     SET_ATM(qnt_tsts,
01270             0.5 * (atm->q[ctl->qnt_tice][ip] + atm->q[ctl->qnt_tnat][ip]));
01271 }
01272 }
01273
01274 /*****
01275
01276 void module_oh_chem(
01277     ctl_t * ctl,
01278     clim_t * clim,
01279     met_t * met0,
01280     met_t * met1,
01281     atm_t * atm,
01282     double *dt) {
01283
01284     /* Set timer... */
01285     SELECT_TIMER("MODULE_OHCHEM", "PHYSICS", NVTX_GPU);
01286
01287     /* Check quantity flags... */
01288     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01289         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01290
01291     const int np = atm->np;
01292 #ifdef _OPENACC
01293 #pragma acc data present(ctl, clim, met0, met1, atm, dt)
01294 #pragma acc parallel loop independent gang vector
01295 #else
01296 #pragma omp parallel for default(shared)
01297 #endif
01298     for (int ip = 0; ip < np; ip++)
01299         if (dt[ip] != 0) {
01300
01301             /* Get temperature... */
01302             double t;
01303             INTPOL_INIT;
01304             INTPOL_3D(t, 1);
01305
01306             /* Use constant reaction rate... */
01307             double k = GSL_NAN;
01308             if (ctl->oh_chem_reaction == 1)
01309                 k = ctl->oh_chem[0];
01310
01311             /* Calculate bimolecular reaction rate... */
01312             else if (ctl->oh_chem_reaction == 2)
01313                 k = ctl->oh_chem[0] * exp(-ctl->oh_chem[1] / t);
01314
01315             /* Calculate termolecular reaction rate... */
01316             if (ctl->oh_chem_reaction == 3) {
01317
01318                 /* Calculate molecular density (IUPAC Data Sheet I.A4.86 SOx15)... */
01319                 double M = 7.243e21 * (atm->p[ip] / 1000.) / t;
01320
01321                 /* Calculate rate coefficient for X + OH + M -> XOH + M
01322                    (JPL Publication 19-05) ... */
01323                 double k0 = ctl->oh_chem[0] *
01324                     (ctl->oh_chem[1] > 0 ? pow(298. / t, ctl->oh_chem[1]) : 1.);
01325                 double ki = ctl->oh_chem[2] *
01326                     (ctl->oh_chem[3] > 0 ? pow(298. / t, ctl->oh_chem[3]) : 1.);
01327                 double c = log10(k0 * M / ki);
01328                 k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01329             }
01330
01331             /* Calculate exponential decay... */
01332             double rate_coef =
01333                 k * clim_oh_diurnal(ctl, clim, atm->time[ip], atm->p[ip],
01334                                     atm->lon[ip],
01335                                     atm->lat[ip]);
01336             double aux = exp(-dt[ip] * rate_coef);
01337             if (ctl->qnt_m >= 0) {
01338                 if (ctl->qnt_mloss_oh >= 0)
01339                     atm->q[ctl->qnt_mloss_oh][ip]

```

```

01340         += atm->q[ctl->qnt_m][ip] * (1 - aux);
01341     atm->q[ctl->qnt_m][ip] *= aux;
01342 }
01343 if (ctl->qnt_vmr >= 0)
01344     atm->q[ctl->qnt_vmr][ip] *= aux;
01345 }
01346 }
01347
01348 /*****
01349
01350 void module_h2o2_chem(
01351     ctl_t * ctl,
01352     clim_t * clim,
01353     met_t * met0,
01354     met_t * met1,
01355     atm_t * atm,
01356     double *dt,
01357     double *rs) {
01358
01359     /* Set timer... */
01360     SELECT_TIMER("MODULE_H2O2CHEM", "PHYSICS", NVTX_GPU);
01361
01362     /* Check quantity flags... */
01363     if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01364         ERRMSG("Module needs quantity mass or volume mixing ratio!");
01365     if (ctl->qnt_vmrimpl < 0)
01366         ERRMSG("Module needs quantity implicit volume mixing ratio!");
01367
01368     /* Create random numbers... */
01369     module_rng(rs, (size_t) atm->np, 0);
01370
01371     const int np = atm->np;
01372     #ifdef _OPENACC
01373     #pragma acc data present (clim,ctl,met0,met1,atm,dt,rs)
01374     #pragma acc parallel loop independent gang vector
01375     #else
01376     #pragma omp parallel for default(shared)
01377     #endif
01378     for (int ip = 0; ip < np; ip++)
01379         if (dt[ip] != 0) {
01380
01381             /* Check whether particle is inside cloud... */
01382             double lwc;
01383             INTPOL_INIT;
01384             INTPOL_3D(lwc, 1);
01385             if (!(lwc > 0))
01386                 continue;
01387
01388             /* Check cloud cover... */
01389             if (rs[ip] > ctl->h2o2_chem_cc)
01390                 continue;
01391
01392             /* Check implicit volume mixing ratio... */
01393             if (atm->q[ctl->qnt_vmrimpl][ip] == 0)
01394                 continue;
01395
01396             /* Get temperature... */
01397             double t;
01398             INTPOL_3D(t, 0);
01399
01400             /* Reaction rate (Berglen et al., 2004)... */
01401             double k = 9.1e7 * exp(-29700 / RI * (1. / t - 1. / 298.15)); // Maass 1999 unit: M^(-2)
01402
01403             /* Henry constant of SO2... */
01404             double H_SO2 = 1.3e-2 * exp(2900 * (1. / t - 1. / 298.15)) * RI * t;
01405             double K_1S = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15)); // unit: M
01406
01407             /* Henry constant of H2O2... */
01408             double H_h2o2 = 8.3e2 * exp(7600 * (1 / t - 1 / 298.15)) * RI * t;
01409
01410             /* Concentration of H2O2 (Barth et al., 1989)... */
01411             double SO2 = atm->q[ctl->qnt_vmrimpl][ip] * 1e9; // vmr unit: ppbv
01412             double h2o2 = H_h2o2
01413                 * clim_h2o2(clim, atm->time[ip], atm->lat[ip], atm->p[ip])
01414                 * 0.59 * exp(-0.687 * SO2) * 1000 / 6.02214e23; // unit: M
01415
01416             /* Volume water content in cloud [m^3 m^(-3)]... */
01417             double rho_air = 100 * atm->p[ip] / (RA * t);
01418             double CWC = lwc * rho_air / 1000;
01419
01420             /* Calculate exponential decay (Rolph et al., 1992)... */
01421             double rate_coef = k * K_1S * h2o2 * H_SO2 * CWC;
01422             double aux = exp(-dt[ip] * rate_coef);
01423             if (ctl->qnt_m >= 0) {
01424                 if (ctl->qnt_mloss_h2o2 >= 0)
01425                     atm->q[ctl->qnt_mloss_h2o2][ip] +=
01426                     atm->q[ctl->qnt_m][ip] * (1 - aux);

```

```

01427         atm->q[ctl->qnt_m][ip] *= aux;
01428     }
01429     if (ctl->qnt_vmr >= 0)
01430         atm->q[ctl->qnt_vmr][ip] *= aux;
01431     }
01432 }
01433
01434 /*****
01435
01436 void module_chemgrid(
01437     ctl_t * ctl,
01438     met_t * met0,
01439     met_t * met1,
01440     atm_t * atm,
01441     double t) {
01442
01443     double *mass, *z, *lon, *lat, *press, *area;
01444
01445     int *ixs, *iys, *izs;
01446
01447     /* Update host... */
01448 #ifdef _OPENACC
01449     SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
01450 #pragma acc update host(atm[:1])
01451 #endif
01452
01453     /* Set timer... */
01454     SELECT_TIMER("MODULE_CHEMGRID", "PHYSICS", NVTX_GPU);
01455
01456     /* Check quantity flags... */
01457     if (ctl->qnt_m < 0)
01458         ERRMSG("Module needs quantity mass!");
01459     if (ctl->qnt_vmrimpl < 0)
01460         ERRMSG("Module needs quantity implicit volume mixing ratio!");
01461
01462     /* Allocate... */
01463     ALLOC(mass, double,
01464         ctl->chemgrid_nx * ctl->chemgrid_ny * ctl->chemgrid_nz);
01465     ALLOC(z, double,
01466         ctl->chemgrid_nz);
01467     ALLOC(lon, double,
01468         ctl->chemgrid_nx);
01469     ALLOC(lat, double,
01470         ctl->chemgrid_ny);
01471     ALLOC(area, double,
01472         ctl->chemgrid_ny);
01473     ALLOC(press, double,
01474         ctl->chemgrid_nz);
01475     ALLOC(ixs, int,
01476         atm->np);
01477     ALLOC(iys, int,
01478         atm->np);
01479     ALLOC(izs, int,
01480         atm->np);
01481
01482     /* Set grid box size... */
01483     double dz = (ctl->chemgrid_z1 - ctl->chemgrid_z0) / ctl->chemgrid_nz;
01484     double dlon = (ctl->chemgrid_lon1 - ctl->chemgrid_lon0) / ctl->chemgrid_nx;
01485     double dlat = (ctl->chemgrid_lat1 - ctl->chemgrid_lat0) / ctl->chemgrid_ny;
01486
01487     /* Set vertical coordinates... */
01488 #pragma omp parallel for default(shared)
01489     for (int iz = 0; iz < ctl->chemgrid_nz; iz++) {
01490         z[iz] = ctl->chemgrid_z0 + dz * (iz + 0.5);
01491         press[iz] = P(z[iz]);
01492     }
01493
01494     /* Set horizontal coordinates... */
01495     for (int ix = 0; ix < ctl->chemgrid_nx; ix++)
01496         lon[ix] = ctl->chemgrid_lon0 + dlon * (ix + 0.5);
01497 #pragma omp parallel for default(shared)
01498     for (int iy = 0; iy < ctl->chemgrid_ny; iy++) {
01499         lat[iy] = ctl->chemgrid_lat0 + dlat * (iy + 0.5);
01500         area[iy] = dlat * dlon * SQR(RE * M_PI / 180.)
01501             * cos(lat[iy] * M_PI / 180.);
01502     }
01503
01504     /* Set time interval for output... */
01505     double t0 = t - 0.5 * ctl->dt_mod;
01506     double t1 = t + 0.5 * ctl->dt_mod;
01507
01508     /* Get indices... */
01509 #pragma omp parallel for default(shared)
01510     for (int ip = 0; ip < atm->np; ip++) {
01511         ixs[ip] = (int) ((atm->lon[ip] - ctl->chemgrid_lon0) / dlon);
01512         iys[ip] = (int) ((atm->lat[ip] - ctl->chemgrid_lat0) / dlat);
01513         izs[ip] = (int) ((Z(atm->p[ip]) - ctl->chemgrid_z0) / dz);

```

```

01514     if (atm->time[ip] < t0 || atm->time[ip] > t1
01515         || ixs[ip] < 0 || ixs[ip] >= ctl->chemgrid_nx
01516         || iys[ip] < 0 || iys[ip] >= ctl->chemgrid_ny
01517         || izs[ip] < 0 || izs[ip] >= ctl->chemgrid_nz)
01518         izs[ip] = -1;
01519     }
01520
01521     /* Average data... */
01522     for (int ip = 0; ip < atm->np; ip++)
01523         if (izs[ip] >= 0)
01524             mass[ARRAY_3D
01525                 (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
01526                 += atm->q[ctl->qnt_m][ip];
01527
01528     /* Interpolate volume mixing ratio... */
01529 #pragma omp parallel for default(shared)
01530     for (int ip = 0; ip < atm->np; ip++)
01531         if (izs[ip] >= 0) {
01532             double temp;
01533             INTPOL_INIT;
01534             intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press[izs[ip]],
01535                               lon[ixs[ip]], lat[iys[ip]], &temp, ci, cw, 1);
01536             atm->q[ctl->qnt_vmrimpl][ip] = MA / ctl->molmass *
01537                 mass[ARRAY_3D
01538                     (ixs[ip], iys[ip], ctl->chemgrid_ny, izs[ip], ctl->chemgrid_nz)]
01539                     / (RHO(press[izs[ip]], temp) * 1e6 * area[iys[ip]] * 1e3 * dz);
01540         }
01541
01542     /* Free... */
01543     free(mass);
01544     free(z);
01545     free(lon);
01546     free(lat);
01547     free(area);
01548     free(press);
01549     free(ixs);
01550     free(iys);
01551     free(izs);
01552
01553     /* Update device... */
01554 #ifdef _OPENACC
01555     SELECT_TIMER("UPDATE_DEVICE", "MEMORY", NVTX_H2D);
01556 #pragma acc update device(atm[:1])
01557 #endif
01558 }
01559
01560 /*****
01561
01562 void module_position(
01563     ctl_t * ctl,
01564     met_t * met0,
01565     met_t * met1,
01566     atm_t * atm,
01567     double *dt) {
01568
01569     /* Set timer... */
01570     SELECT_TIMER("MODULE_POSITION", "PHYSICS", NVTX_GPU);
01571
01572     const int np = atm->np;
01573 #ifdef _OPENACC
01574 #pragma acc data present(met0, met1, atm, dt)
01575 #pragma acc parallel loop independent gang vector
01576 #else
01577 #pragma omp parallel for default(shared)
01578 #endif
01579     for (int ip = 0; ip < np; ip++)
01580         if (dt[ip] != 0) {
01581
01582             /* Init... */
01583             double ps;
01584             INTPOL_INIT;
01585
01586             /* Calculate modulo... */
01587             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01588             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01589
01590             /* Check latitude... */
01591             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01592                 if (atm->lat[ip] > 90) {
01593                     atm->lat[ip] = 180 - atm->lat[ip];
01594                     atm->lon[ip] += 180;
01595                 }
01596                 if (atm->lat[ip] < -90) {
01597                     atm->lat[ip] = -180 - atm->lat[ip];
01598                     atm->lon[ip] += 180;
01599                 }
01600             }

```

```

01601
01602     /* Check longitude... */
01603     while (atm->lon[ip] < -180)
01604         atm->lon[ip] += 360;
01605     while (atm->lon[ip] >= 180)
01606         atm->lon[ip] -= 360;
01607
01608     /* Check pressure... */
01609     if (atm->p[ip] < met0->p[met0->np - 1]) {
01610         if (ctl->reflect)
01611             atm->p[ip] = 2. * met0->p[met0->np - 1] - atm->p[ip];
01612         else
01613             atm->p[ip] = met0->p[met0->np - 1];
01614     } else if (atm->p[ip] > 300.) {
01615         INTPOL_2D(ps, 1);
01616         if (atm->p[ip] > ps) {
01617             if (ctl->reflect)
01618                 atm->p[ip] = 2. * ps - atm->p[ip];
01619             else
01620                 atm->p[ip] = ps;
01621         }
01622     }
01623 }
01624 }
01625
01626 /*****
01627
01628 void module_rng_init(
01629     int ntask) {
01630
01631     /* Initialize random number generator... */
01632     #ifdef _OPENACC
01633
01634         if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT) !=
01635             CURAND_STATUS_SUCCESS)
01636             ERRMSG("Cannot create random number generator!");
01637         if (curandSetPseudoRandomGeneratorSeed(rng, ntask) != CURAND_STATUS_SUCCESS)
01638             ERRMSG("Cannot set seed for random number generator!");
01639         if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
01640             != CURAND_STATUS_SUCCESS)
01641             ERRMSG("Cannot set stream for random number generator!");
01642
01643     #else
01644         gsl_rng_env_setup();
01645         if (omp_get_max_threads() > NTHREADS)
01646             ERRMSG("Too many threads!");
01647         for (int i = 0; i < NTHREADS; i++) {
01648             rng[i] = gsl_rng_alloc(gsl_rng_default);
01649             gsl_rng_set(rng[i],
01650                 gsl_rng_default_seed + (long unsigned) (ntask * NTHREADS +
01651                     i));
01652         }
01653     #endif
01654 }
01655 #endif
01656 }
01657
01658 /*****
01659
01660 void module_rng(
01661     double *rs,
01662     size_t n,
01663     int method) {
01664
01665     #ifdef _OPENACC
01666     #pragma acc host_data use_device(rs)
01667     {
01668         /* Uniform distribution... */
01669         if (method == 0) {
01670             if (curandGenerateUniformDouble(rng, rs, (n < 4 ? 4 : n)) !=
01671                 CURAND_STATUS_SUCCESS)
01672                 ERRMSG("Cannot create random numbers!");
01673         }
01674
01675         /* Normal distribution... */
01676         else if (method == 1) {
01677             if (curandGenerateNormalDouble(rng, rs, (n < 4 ? 4 : n), 0.0, 1.0) !=
01678                 CURAND_STATUS_SUCCESS)
01679                 ERRMSG("Cannot create random numbers!");
01680         }
01681     }
01682     #endif
01683
01684     #else
01685         /* Uniform distribution... */
01686         if (method == 0) {

```

```

01688 #pragma omp parallel for default(shared)
01689     for (size_t i = 0; i < n; ++i)
01690         rs[i] = gsl_rng_uniform(rng[omp_get_thread_num()]);
01691     }
01692
01693     /* Normal distribution... */
01694     else if (method == 1) {
01695 #pragma omp parallel for default(shared)
01696         for (size_t i = 0; i < n; ++i)
01697             rs[i] = gsl_rng_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
01698     }
01699 #endif
01700 }
01701
01702 /*****
01703
01704 void module_sedi(
01705     ctl_t * ctl,
01706     met_t * met0,
01707     met_t * met1,
01708     atm_t * atm,
01709     double *dt) {
01710
01711     /* Set timer... */
01712     SELECT_TIMER("MODULE_SEDI", "PHYSICS", NVTX_GPU);
01713
01714     const int np = atm->np;
01715 #ifdef _OPENACC
01716 #pragma acc data present(ctl, met0, met1, atm, dt)
01717 #pragma acc parallel loop independent gang vector
01718 #else
01719 #pragma omp parallel for default(shared)
01720 #endif
01721     for (int ip = 0; ip < np; ip++)
01722         if (dt[ip] != 0) {
01723
01724             /* Get temperature... */
01725             double t;
01726             INTPOL_INIT;
01727             INTPOL_3D(t, 1);
01728
01729             /* Sedimentation velocity... */
01730             double v_s = sedi(atm->p[ip], t, atm->q[ctl->qnt_rp][ip],
01731                             atm->q[ctl->qnt_rhop][ip]);
01732
01733             /* Calculate pressure change... */
01734             atm->p[ip] += DZ2DP(v_s * dt[ip] / 1000., atm->p[ip]);
01735         }
01736 }
01737
01738 /*****
01739
01740 void module_sort(
01741     ctl_t * ctl,
01742     met_t * met0,
01743     atm_t * atm) {
01744
01745     /* Set timer... */
01746     SELECT_TIMER("MODULE_SORT_BOXINDEX", "MEMORY", NVTX_GPU);
01747
01748     /* Allocate... */
01749     const int np = atm->np;
01750     double *restrict const a = (double *) malloc((size_t) np * sizeof(double));
01751     int *restrict const p = (int *) malloc((size_t) np * sizeof(int));
01752
01753 #ifdef _OPENACC
01754 #pragma acc enter data create(a[0:np], p[0:np])
01755 #pragma acc data present(ctl, met0, atm, a, p)
01756 #endif
01757
01758     /* Get box index... */
01759 #ifdef _OPENACC
01760 #pragma acc parallel loop independent gang vector
01761 #else
01762 #pragma omp parallel for default(shared)
01763 #endif
01764     for (int ip = 0; ip < np; ip++) {
01765         a[ip] =
01766             (double) ((locate_reg(met0->lon, met0->nx, atm->lon[ip]) * met0->ny +
01767                     locate_reg(met0->lat, met0->ny,
01768                             atm->lat[ip])) * met0->np + locate_irr(met0->p,
01769                             met0->np,
01770                             atm->p
01771                             [ip]));
01772         p[ip] = ip;
01773     }
01774

```

```

01775  /* Set timer... */
01776  SELECT_TIMER("MODULE_SORT_THRUST", "MEMORY", NVTX_GPU);
01777
01778  /* Sorting... */
01779  #ifdef _OPENACC
01780  {
01781  #ifdef THRUST
01782  {
01783  #pragma acc host_data use_device(a, p)
01784      thrustSortWrapper(a, np, p);
01785  }
01786  #else
01787  {
01788  #pragma acc update host(a[0:np], p[0:np])
01789  #pragma omp parallel
01790  {
01791  #pragma omp single nowait
01792      quicksort(a, p, 0, np - 1);
01793  }
01794  #pragma acc update device(a[0:np], p[0:np])
01795  }
01796  #endif
01797  }
01798  #else
01799  {
01800  #ifdef THRUST
01801  {
01802      thrustSortWrapper(a, np, p);
01803  }
01804  #else
01805  {
01806  #pragma omp parallel
01807  {
01808  #pragma omp single nowait
01809      quicksort(a, p, 0, np - 1);
01810  }
01811  }
01812  #endif
01813  }
01814  #endif
01815
01816  /* Set timer... */
01817  SELECT_TIMER("MODULE_SORT_REORDERING", "MEMORY", NVTX_GPU);
01818
01819  /* Sort data... */
01820  module_sort_help(atm->time, p, np);
01821  module_sort_help(atm->p, p, np);
01822  module_sort_help(atm->lon, p, np);
01823  module_sort_help(atm->lat, p, np);
01824  for (int iq = 0; iq < atm->nq; iq++)
01825      module_sort_help(atm->q[iq], p, np);
01826
01827  /* Free... */
01828  #ifdef _OPENACC
01829  #pragma acc exit data delete(a,p)
01830  #endif
01831  free(a);
01832  free(p);
01833  }
01834
01835  /*****
01836
01837  void module_sort_help(
01838      double *a,
01839      int *p,
01840      int np) {
01841
01842      /* Allocate... */
01843      double *restrict const help =
01844          (double *) malloc((size_t) np * sizeof(double));
01845
01846      /* Reordering of array... */
01847  #ifdef _OPENACC
01848  #pragma acc enter data create(help[0:np])
01849  #pragma acc data present(a,p,help)
01850  #pragma acc parallel loop independent gang vector
01851  #endif
01852      for (int ip = 0; ip < np; ip++)
01853          help[ip] = a[p[ip]];
01854  #ifdef _OPENACC
01855  #pragma acc parallel loop independent gang vector
01856  #endif
01857      for (int ip = 0; ip < np; ip++)
01858          a[ip] = help[ip];
01859
01860      /* Free... */
01861  #ifdef _OPENACC

```



```

01862 #pragma acc exit data delete(help)
01863 #endif
01864     free(help);
01865 }
01866
01867 /*****
01868
01869 void module_timesteps(
01870     ctl_t * ctl,
01871     atm_t * atm,
01872     met_t * met0,
01873     double *dt,
01874     double t) {
01875
01876     /* Set timer... */
01877     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01878
01879     const double latmin = gsl_stats_min(met0->lat, 1, (size_t) met0->ny),
01880         latmax = gsl_stats_max(met0->lat, 1, (size_t) met0->ny);
01881
01882     const int np = atm->np,
01883         local = (fabs(met0->lon[met0->nx - 1] - met0->lon[0] - 360.0) >= 0.01);
01884
01885 #ifdef _OPENACC
01886 #pragma acc data present(ctl, atm, dt)
01887 #pragma acc parallel loop independent gang vector
01888 #else
01889 #pragma omp parallel for default(shared)
01890 #endif
01891     for (int ip = 0; ip < np; ip++) {
01892
01893         /* Set time step for each air parcel... */
01894         if ((ctl->direction * (atm->time[ip] - ctl->t_start) >= 0
01895             && ctl->direction * (atm->time[ip] - ctl->t_stop) <= 0
01896             && ctl->direction * (atm->time[ip] - t) < 0))
01897             dt[ip] = t - atm->time[ip];
01898         else
01899             dt[ip] = 0.0;
01900
01901         /* Check horizontal boundaries of local meteo data... */
01902         if (local && (atm->lon[ip] <= met0->lon[0]
01903             || atm->lon[ip] >= met0->lon[met0->nx - 1]
01904             || atm->lat[ip] <= latmin || atm->lat[ip] >= latmax))
01905             dt[ip] = 0.0;
01906     }
01907 }
01908
01909 /*****
01910
01911 void module_timesteps_init(
01912     ctl_t * ctl,
01913     atm_t * atm) {
01914
01915     /* Set timer... */
01916     SELECT_TIMER("MODULE_TIMESTEPS", "PHYSICS", NVTX_GPU);
01917
01918     /* Set start time... */
01919     if (ctl->direction == 1) {
01920         ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01921         if (ctl->t_stop > 1e99)
01922             ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01923     } else {
01924         ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
01925         if (ctl->t_stop > 1e99)
01926             ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
01927     }
01928
01929     /* Check time interval... */
01930     if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
01931         ERRMSG("Nothing to do! Check T_STOP and DIRECTION!");
01932
01933     /* Round start time... */
01934     if (ctl->direction == 1)
01935         ctl->t_start = floor(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01936     else
01937         ctl->t_start = ceil(ctl->t_start / ctl->dt_mod) * ctl->dt_mod;
01938 }
01939
01940 /*****
01941
01942 void module_wet_deposition(
01943     ctl_t * ctl,
01944     met_t * met0,
01945     met_t * met1,
01946     atm_t * atm,
01947     double *dt) {
01948

```

```

01949  /* Set timer... */
01950  SELECT_TIMER("MODULE_WETDEPO", "PHYSICS", NVTX_GPU);
01951
01952  /* Check quantity flags... */
01953  if (ctl->qnt_m < 0 && ctl->qnt_vmr < 0)
01954      ERRMSG("Module needs quantity mass or volume mixing ratio!");
01955
01956  const int np = atm->np;
01957  #ifdef _OPENACC
01958  #pragma acc data present(ctl, met0, met1, atm, dt)
01959  #pragma acc parallel loop independent gang vector
01960  #else
01961  #pragma omp parallel for default(shared)
01962  #endif
01963  for (int ip = 0; ip < np; ip++)
01964      if (dt[ip] != 0) {
01965
01966          double cl, dz, h, lambda = 0, t, iwc, lwc, pct, pcb;
01967
01968          /* Check whether particle is below cloud top... */
01969          INTPOL_INIT;
01970          INTPOL_2D(pct, 1);
01971          if (!isfinite(pct) || atm->p[ip] <= pct)
01972              continue;
01973
01974          /* Get cloud bottom pressure... */
01975          INTPOL_2D(pcb, 0);
01976
01977          /* Estimate precipitation rate (Pisso et al., 2019)... */
01978          INTPOL_2D(cl, 0);
01979          double Is =
01980              pow(1. / ctl->wet_depo_pre[0] * cl, 1. / ctl->wet_depo_pre[1]);
01981          if (Is < 0.01)
01982              continue;
01983
01984          /* Check whether particle is inside or below cloud... */
01985          INTPOL_3D(lwc, 1);
01986          INTPOL_3D(iwc, 0);
01987          int inside = (iwc > 0 || lwc > 0);
01988
01989          /* Get temperature... */
01990          INTPOL_3D(t, 0);
01991
01992          /* Calculate in-cloud scavenging coefficient... */
01993          if (inside) {
01994
01995              /* Calculate retention factor... */
01996              double eta;
01997              if (t > 273.15)
01998                  eta = 1;
01999              else if (t <= 238.15)
02000                  eta = ctl->wet_depo_ic_ret_ratio;
02001              else
02002                  eta = LIN(273.15, 1, 238.15, ctl->wet_depo_ic_ret_ratio, t);
02003
02004              /* Use exponential dependency for particles ... */
02005              if (ctl->wet_depo_ic_a > 0)
02006                  lambda = ctl->wet_depo_ic_a * pow(Is, ctl->wet_depo_ic_b) * eta;
02007
02008              /* Use Henry's law for gases... */
02009              else if (ctl->wet_depo_ic_h[0] > 0) {
02010
02011                  /* Get Henry's constant (Sander, 2015)... */
02012                  h = ctl->wet_depo_ic_h[0]
02013                      * exp(ctl->wet_depo_ic_h[1] * (1. / t - 1. / 298.15));
02014
02015                  /* Use effective Henry's constant for SO2
02016                     (Berglen, 2004; Simpson, 2012)... */
02017                  if (ctl->wet_depo_ic_h[2] > 0) {
02018                      double H_ion = pow(10, ctl->wet_depo_ic_h[2] * (-1));
02019                      double K_1 = 1.23e-2 * exp(2.01e3 * (1. / t - 1. / 298.15));
02020                      double K_2 = 6e-8 * exp(1.12e3 * (1. / t - 1. / 298.15));
02021                      h *= (1 + K_1 / H_ion + K_1 * K_2 / pow(H_ion, 2));
02022                  }
02023
02024                  /* Estimate depth of cloud layer... */
02025                  dz = 1e3 * (Z(pct) - Z(pcb));
02026
02027                  /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
02028                  lambda = h * RI * t * Is / 3.6e6 / dz * eta;
02029              }
02030          }
02031
02032          /* Calculate below-cloud scavenging coefficient... */
02033          else {
02034
02035              /* Calculate retention factor... */

```

```

02036     double eta;
02037     if (t > 270)
02038         eta = 1;
02039     else
02040         eta = ctl->wet_depo_bc_ret_ratio;
02041
02042     /* Use exponential dependency for particles... */
02043     if (ctl->wet_depo_bc_a > 0)
02044         lambda = ctl->wet_depo_bc_a * pow(Is, ctl->wet_depo_bc_b) * eta;
02045
02046     /* Use Henry's law for gases... */
02047     else if (ctl->wet_depo_bc_h[0] > 0) {
02048
02049         /* Get Henry's constant (Sander, 2015)... */
02050         h = ctl->wet_depo_bc_h[0]
02051             * exp(ctl->wet_depo_bc_h[1] * (1. / t - 1. / 298.15));
02052
02053         /* Estimate depth of cloud layer... */
02054         dz = 1e3 * (Z(pct) - Z(pcb));
02055
02056         /* Calculate scavenging coefficient (Draxler and Hess, 1997)... */
02057         lambda = h * RI * t * Is / 3.6e6 / dz * eta;
02058     }
02059 }
02060
02061 /* Calculate exponential decay of mass... */
02062 double aux = exp(-dt[ip] * lambda);
02063 if (ctl->qnt_m >= 0) {
02064     if (ctl->qnt_mloss_wet >= 0)
02065         atm->q[ctl->qnt_mloss_wet][ip]
02066             += atm->q[ctl->qnt_m][ip] * (1 - aux);
02067     atm->q[ctl->qnt_m][ip] *= aux;
02068 }
02069 if (ctl->qnt_vmr >= 0)
02070     atm->q[ctl->qnt_vmr][ip] *= aux;
02071 }
02072 }
02073
02074 /*****
02075
02076 void write_output(
02077     const char *dirname,
02078     ctl_t * ctl,
02079     met_t * met0,
02080     met_t * met1,
02081     atm_t * atm,
02082     double t) {
02083
02084     char ext[10], filename[2 * LEN];
02085
02086     double r;
02087
02088     int year, mon, day, hour, min, sec;
02089
02090     /* Get time... */
02091     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
02092
02093     /* Update host... */
02094 #ifdef _OPENACC
02095     if ((ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0)
02096         || (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0)
02097         || (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0)
02098         || (ctl->csi_basename[0] != '-' || ctl->prof_basename[0] != '-'
02099             || ctl->sample_basename[0] != '-' || ctl->stat_basename[0] != '-') {
02100         SELECT_TIMER("UPDATE_HOST", "MEMORY", NVTX_D2H);
02101 #pragma acc update host(atm[:1])
02102     }
02103 #endif
02104
02105     /* Write atmospheric data... */
02106     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
02107         if (ctl->atm_type == 0)
02108             sprintf(ext, "tab");
02109         else if (ctl->atm_type == 1)
02110             sprintf(ext, "bin");
02111         else if (ctl->atm_type == 2)
02112             sprintf(ext, "nc");
02113         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.%s",
02114             dirname, ctl->atm_basename, year, mon, day, hour, min, ext);
02115         write_atm(filename, ctl, atm, t);
02116     }
02117
02118     /* Write gridded data... */
02119     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
02120         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.%s",
02121             dirname, ctl->grid_basename, year, mon, day, hour, min,
02122             ctl->grid_type == 0 ? "tab" : "nc");

```

```

02123     write_grid(filename, ctl, met0, met1, atm, t);
02124 }
02125
02126 /* Write CSI data... */
02127 if (ctl->csi_basename[0] != '-') {
02128     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
02129     write_csi(filename, ctl, atm, t);
02130 }
02131
02132 /* Write ensemble data... */
02133 if (ctl->ens_basename[0] != '-' && fmod(t, ctl->ens_dt_out) == 0) {
02134     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
02135             dirname, ctl->ens_basename, year, mon, day, hour, min);
02136     write_ens(filename, ctl, atm, t);
02137 }
02138
02139 /* Write profile data... */
02140 if (ctl->prof_basename[0] != '-') {
02141     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
02142     write_prof(filename, ctl, met0, met1, atm, t);
02143 }
02144
02145 /* Write sample data... */
02146 if (ctl->sample_basename[0] != '-') {
02147     sprintf(filename, "%s/%s.tab", dirname, ctl->sample_basename);
02148     write_sample(filename, ctl, met0, met1, atm, t);
02149 }
02150
02151 /* Write station data... */
02152 if (ctl->stat_basename[0] != '-') {
02153     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
02154     write_station(filename, ctl, atm, t);
02155 }
02156 }

```

5.49 tropo.c File Reference

Create tropopause data set from meteorological data.

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

Functions

- void [get_tropo](#) (int met_tropo, [ctl_t](#) *ctl, [clim_t](#) *clim, [met_t](#) *met, double *lons, int nx, double *lats, int ny, double *pt, double *zt, double *tt, double *qt, double *o3t, double *ps, double *zs)
- int [main](#) (int argc, char *argv[])

5.49.1 Detailed Description

Create tropopause data set from meteorological data.

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

5.49.2 Function Documentation

```

5.49.2.1 get_tropo() void get_tropo (
    int met_tropo,
    ctl_t * ctl,
    clim_t * clim,
    met_t * met,
    double * lons,
    int nx,
    double * lats,
    int ny,
    double * pt,
    double * zt,
    double * tt,
    double * qt,
    double * o3t,
    double * ps,
    double * zs )

```

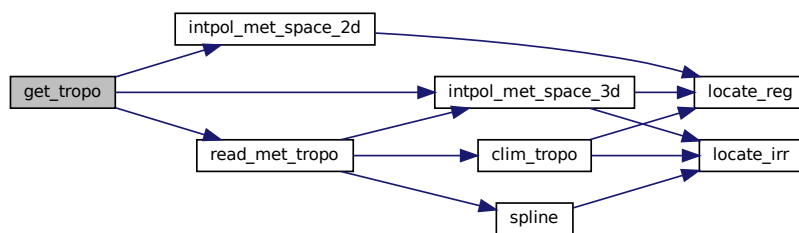
Definition at line 279 of file [tropo.c](#).

```

00294     {
00295
00296     INTPOL_INIT;
00297
00298     ctl->met_tropo = met_tropo;
00299     read_met_tropo(ctl, clim, met);
00300     #pragma omp parallel for default(shared) private(ci,cw)
00301     for (int ix = 0; ix < nx; ix++)
00302         for (int iy = 0; iy < ny; iy++) {
00303         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00304                             &pt[iy * nx + ix], ci, cw, 1);
00305         intpol_met_space_2d(met, met->ps, lons[ix], lats[iy],
00306                             &ps[iy * nx + ix], ci, cw, 0);
00307         intpol_met_space_2d(met, met->zs, lons[ix], lats[iy],
00308                             &zs[iy * nx + ix], ci, cw, 0);
00309         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00310                             lats[iy], &z_t[iy * nx + ix], ci, cw, 1);
00311         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00312                             lats[iy], &t_t[iy * nx + ix], ci, cw, 0);
00313         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00314                             lats[iy], &q_t[iy * nx + ix], ci, cw, 0);
00315         intpol_met_space_3d(met, met->o3, pt[iy * nx + ix], lons[ix],
00316                             lats[iy], &o3_t[iy * nx + ix], ci, cw, 0);
00317     }
00318 }

```

Here is the call graph for this function:



```

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

```

Definition at line 52 of file `tropo.c`.

```

00054     {
00055
00056     ctl_t ctl;
00057
00058     clim_t *clim;
00059
00060     met_t *met;
00061
00062     static double ps[EX * EY], pt[EX * EY], qt[EX * EY], o3t[EX * EY],
00063         zs[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0, lon1, lons[EX], dlon,
00064         lat, lat0, lat1, lats[EY], dlat;
00065
00066     static int init, i, nx, ny, nt, ncid, varid, dims[3], h2o, o3;
00067
00068     static size_t count[10], start[10];
00069
00070     /* Allocate... */
00071     ALLOC(clim, clim_t, 1);
00072     ALLOC(met, met_t, 1);
00073
00074     /* Check arguments... */
00075     if (argc < 4)
00076         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00077
00078     /* Read control parameters... */
00079     read_ctl(argv[1], argc, argv, &ctl);
00080     lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00081     lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00082     dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00083     lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00084     lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00085     dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00086     h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00087     o3 = (int) scan_ctl(argv[1], argc, argv, "TROPO_O3", -1, "1", NULL);
00088
00089     /* Read climatological data... */
00090     read_clim(&ctl, clim);
00091
00092     /* Loop over files... */
00093     for (i = 3; i < argc; i++) {
00094
00095         /* Read meteorological data... */
00096         ctl.met_tropo = 0;
00097         if (!read_met(argv[i], &ctl, clim, met))
00098             continue;
00099
00100         /* Set horizontal grid... */
00101         if (!init) {
00102             init = 1;
00103
00104             /* Get grid... */
00105             if (dlon <= 0)
00106                 dlon = fabs(met->lon[1] - met->lon[0]);
00107             if (dlat <= 0)
00108                 dlat = fabs(met->lat[1] - met->lat[0]);
00109             if (lon0 < -360 && lon1 > 360) {
00110                 lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00111                 lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00112             }
00113             nx = ny = 0;
00114             for (lon = lon0; lon <= lon1; lon += dlon) {
00115                 lons[nx] = lon;
00116                 if ((++nx) > EX)
00117                     ERRMSG("Too many longitudes!");
00118             }
00119             if (lat0 < -90 && lat1 > 90) {
00120                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00121                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00122             }
00123             for (lat = lat0; lat <= lat1; lat += dlat) {
00124                 lats[ny] = lat;
00125                 if ((++ny) > EY)
00126                     ERRMSG("Too many latitudes!");
00127             }
00128
00129             /* Create netCDF file... */
00130             LOG(1, "Write tropopause data file: %s", argv[2]);
00131             NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00132
00133             /* Create dimensions... */
00134             NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00135             NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00136             NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00137
00138             /* Create variables... */
00139             NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time",

```

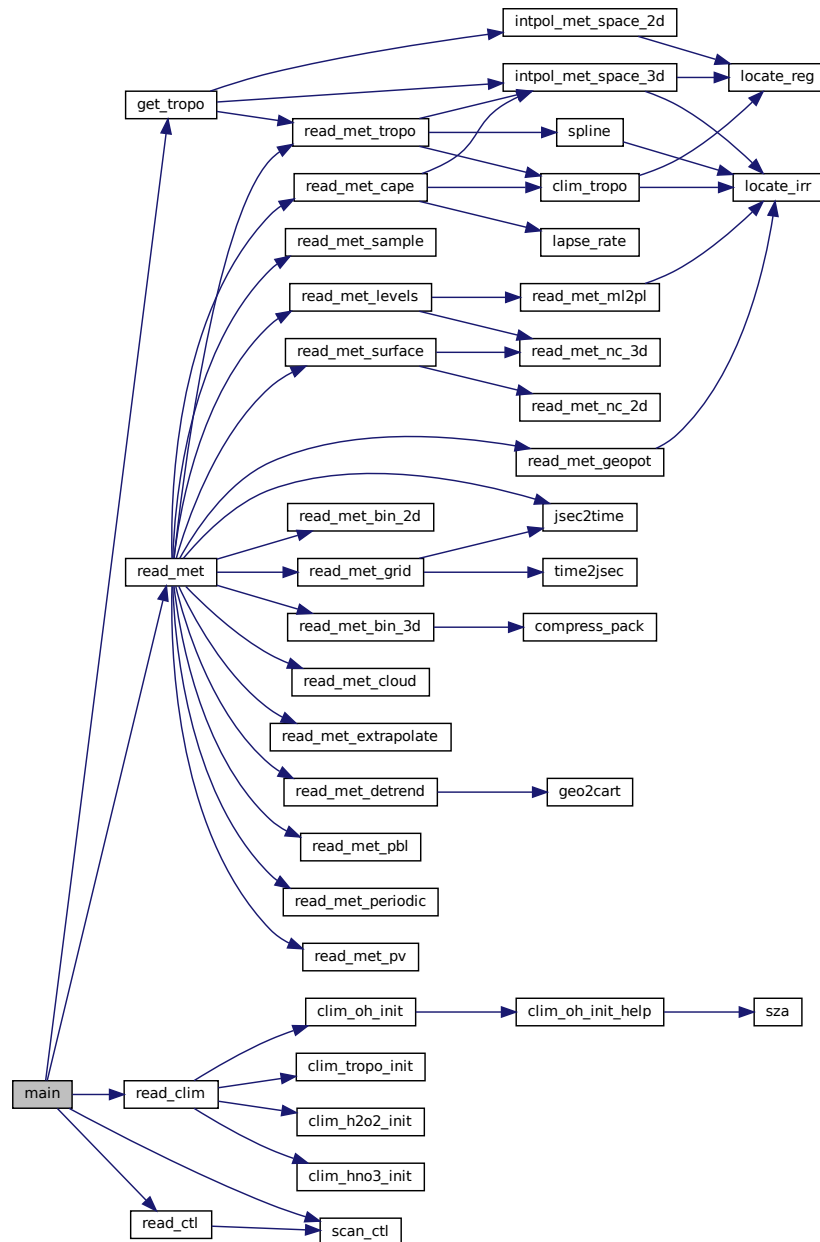
```

00140         "seconds since 2000-01-01 00:00:00 UTC");
00141     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[1], "latitude", "degrees_north");
00142     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[2], "longitude", "degrees_east");
00143
00144     NC_DEF_VAR("clp_z", NC_FLOAT, 3, &dims[0], "cold point height", "km");
00145     NC_DEF_VAR("clp_p", NC_FLOAT, 3, &dims[0], "cold point pressure",
00146               "hPa");
00147     NC_DEF_VAR("clp_t", NC_FLOAT, 3, &dims[0], "cold point temperature",
00148               "K");
00149     if (h2o)
00150         NC_DEF_VAR("clp_q", NC_FLOAT, 3, &dims[0], "cold point water vapor",
00151               "ppv");
00152     if (o3)
00153         NC_DEF_VAR("clp_o3", NC_FLOAT, 3, &dims[0], "cold point ozone",
00154               "ppv");
00155
00156     NC_DEF_VAR("dyn_z", NC_FLOAT, 3, &dims[0],
00157               "dynamical tropopause height", "km");
00158     NC_DEF_VAR("dyn_p", NC_FLOAT, 3, &dims[0],
00159               "dynamical tropopause pressure", "hPa");
00160     NC_DEF_VAR("dyn_t", NC_FLOAT, 3, &dims[0],
00161               "dynamical tropopause temperature", "K");
00162     if (h2o)
00163         NC_DEF_VAR("dyn_q", NC_FLOAT, 3, &dims[0],
00164               "dynamical tropopause water vapor", "ppv");
00165     if (o3)
00166         NC_DEF_VAR("dyn_o3", NC_FLOAT, 3, &dims[0],
00167               "dynamical tropopause ozone", "ppv");
00168
00169     NC_DEF_VAR("wmo_1st_z", NC_FLOAT, 3, &dims[0],
00170               "WMO 1st tropopause height", "km");
00171     NC_DEF_VAR("wmo_1st_p", NC_FLOAT, 3, &dims[0],
00172               "WMO 1st tropopause pressure", "hPa");
00173     NC_DEF_VAR("wmo_1st_t", NC_FLOAT, 3, &dims[0],
00174               "WMO 1st tropopause temperature", "K");
00175     if (h2o)
00176         NC_DEF_VAR("wmo_1st_q", NC_FLOAT, 3, &dims[0],
00177               "WMO 1st tropopause water vapor", "ppv");
00178     if (o3)
00179         NC_DEF_VAR("wmo_1st_o3", NC_FLOAT, 3, &dims[0],
00180               "WMO 1st tropopause ozone", "ppv");
00181
00182     NC_DEF_VAR("wmo_2nd_z", NC_FLOAT, 3, &dims[0],
00183               "WMO 2nd tropopause height", "km");
00184     NC_DEF_VAR("wmo_2nd_p", NC_FLOAT, 3, &dims[0],
00185               "WMO 2nd tropopause pressure", "hPa");
00186     NC_DEF_VAR("wmo_2nd_t", NC_FLOAT, 3, &dims[0],
00187               "WMO 2nd tropopause temperature", "K");
00188     if (h2o)
00189         NC_DEF_VAR("wmo_2nd_q", NC_FLOAT, 3, &dims[0],
00190               "WMO 2nd tropopause water vapor", "ppv");
00191     if (o3)
00192         NC_DEF_VAR("wmo_2nd_o3", NC_FLOAT, 3, &dims[0],
00193               "WMO 2nd tropopause ozone", "ppv");
00194
00195     NC_DEF_VAR("ps", NC_FLOAT, 3, &dims[0], "surface pressure", "hPa");
00196     NC_DEF_VAR("zs", NC_FLOAT, 3, &dims[0], "surface height", "km");
00197
00198     /* End definition... */
00199     NC(nc_enddef(ncid));
00200
00201     /* Write longitude and latitude... */
00202     NC_PUT_DOUBLE("lat", lats, 0);
00203     NC_PUT_DOUBLE("lon", lons, 0);
00204 }
00205
00206     /* Write time... */
00207     start[0] = (size_t) nt;
00208     count[0] = 1;
00209     start[1] = 0;
00210     count[1] = (size_t) ny;
00211     start[2] = 0;
00212     count[2] = (size_t) nx;
00213     NC_PUT_DOUBLE("time", &met->time, 1);
00214
00215     /* Get cold point... */
00216     get_tropo(2, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00217             zs);
00218     NC_PUT_DOUBLE("clp_z", zt, 1);
00219     NC_PUT_DOUBLE("clp_p", pt, 1);
00220     NC_PUT_DOUBLE("clp_t", tt, 1);
00221     if (h2o)
00222         NC_PUT_DOUBLE("clp_q", qt, 1);
00223     if (o3)
00224         NC_PUT_DOUBLE("clp_o3", o3t, 1);
00225
00226     /* Get dynamical tropopause... */

```

```
00227     get_tropo(5, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00228               zs);
00229     NC_PUT_DOUBLE("dyn_z", zt, 1);
00230     NC_PUT_DOUBLE("dyn_p", pt, 1);
00231     NC_PUT_DOUBLE("dyn_t", tt, 1);
00232     if (h2o)
00233         NC_PUT_DOUBLE("dyn_q", qt, 1);
00234     if (o3)
00235         NC_PUT_DOUBLE("dyn_o3", o3t, 1);
00236
00237     /* Get WMO 1st tropopause... */
00238     get_tropo(3, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00239               zs);
00240     NC_PUT_DOUBLE("wmo_1st_z", zt, 1);
00241     NC_PUT_DOUBLE("wmo_1st_p", pt, 1);
00242     NC_PUT_DOUBLE("wmo_1st_t", tt, 1);
00243     if (h2o)
00244         NC_PUT_DOUBLE("wmo_1st_q", qt, 1);
00245     if (o3)
00246         NC_PUT_DOUBLE("wmo_1st_o3", o3t, 1);
00247
00248     /* Get WMO 2nd tropopause... */
00249     get_tropo(4, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00250               zs);
00251     NC_PUT_DOUBLE("wmo_2nd_z", zt, 1);
00252     NC_PUT_DOUBLE("wmo_2nd_p", pt, 1);
00253     NC_PUT_DOUBLE("wmo_2nd_t", tt, 1);
00254     if (h2o)
00255         NC_PUT_DOUBLE("wmo_2nd_q", qt, 1);
00256     if (o3)
00257         NC_PUT_DOUBLE("wmo_2nd_o3", o3t, 1);
00258
00259     /* Write surface data... */
00260     NC_PUT_DOUBLE("ps", ps, 1);
00261     NC_PUT_DOUBLE("zs", zs, 1);
00262
00263     /* Increment time step counter... */
00264     nt++;
00265 }
00266
00267 /* Close file... */
00268 NC(nc_close(ncid));
00269
00270 /* Free... */
00271 free(clim);
00272 free(met);
00273
00274 return EXIT_SUCCESS;
00275 }
```


Here is the call graph for this function:



5.50 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) 2013-2023 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  /* -----
00028      Functions...
00029  ----- */
00030
00031  void get_tropo(
00032      int met_tropo,
00033      ctl_t * ctl,
00034      clim_t * clim,
00035      met_t * met,
00036      double *lons,
00037      int nx,
00038      double *lats,
00039      int ny,
00040      double *pt,
00041      double *zt,
00042      double *tt,
00043      double *qt,
00044      double *o3t,
00045      double *ps,
00046      double *zs);
00047
00048  /* -----
00049      Main...
00050  ----- */
00051
00052  int main(
00053      int argc,
00054      char *argv[]) {
00055
00056      ctl_t ctl;
00057
00058      clim_t *clim;
00059
00060      met_t *met;
00061
00062      static double ps[EX * EY], pt[EX * EY], qt[EX * EY], o3t[EX * EY],
00063          zs[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0, lon1, lons[EX], dlon,
00064          lat, lat0, lat1, lats[EY], dlat;
00065
00066      static int init, i, nx, ny, nt, ncid, varid, dims[3], h2o, o3;
00067
00068      static size_t count[10], start[10];
00069
00070      /* Allocate... */
00071      ALLOC(clim, clim_t, 1);
00072      ALLOC(met, met_t, 1);
00073
00074      /* Check arguments... */
00075      if (argc < 4)
00076          ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00077
00078      /* Read control parameters... */
00079      read_ctl(argv[1], argc, argv, &ctl);
00080      lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00081      lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00082      dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00083      lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00084      lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00085      dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00086      h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00087      o3 = (int) scan_ctl(argv[1], argc, argv, "TROPO_O3", -1, "1", NULL);
00088
00089      /* Read climatological data... */
00090      read_clim(&ctl, clim);
00091
00092      /* Loop over files... */
00093      for (i = 3; i < argc; i++) {
00094
00095          /* Read meteorological data... */
00096          ctl.met_tropo = 0;
00097          if (!read_met(argv[i], &ctl, clim, met))
00098              continue;
00099
00100          /* Set horizontal grid... */
00101          if (!init) {
00102              init = 1;
00103
00104              /* Get grid... */
00105              if (dlon <= 0)
00106                  dlon = fabs(met->lon[1] - met->lon[0]);

```

```

00107     if (dlat <= 0)
00108         dlat = fabs(met->lat[1] - met->lat[0]);
00109     if (lon0 < -360 && lon1 > 360) {
00110         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00111         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00112     }
00113     nx = ny = 0;
00114     for (lon = lon0; lon <= lon1; lon += dlon) {
00115         lons[nx] = lon;
00116         if ((++nx) > EX)
00117             ERRMSG("Too many longitudes!");
00118     }
00119     if (lat0 < -90 && lat1 > 90) {
00120         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00121         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00122     }
00123     for (lat = lat0; lat <= lat1; lat += dlat) {
00124         lats[ny] = lat;
00125         if ((++ny) > EY)
00126             ERRMSG("Too many latitudes!");
00127     }
00128
00129     /* Create netCDF file... */
00130     LOG(1, "Write tropopause data file: %s", argv[2]);
00131     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00132
00133     /* Create dimensions... */
00134     NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00135     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00136     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00137
00138     /* Create variables... */
00139     NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time",
00140               "seconds since 2000-01-01 00:00:00 UTC");
00141     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[1], "latitude", "degrees_north");
00142     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[2], "longitude", "degrees_east");
00143
00144     NC_DEF_VAR("clp_z", NC_FLOAT, 3, &dims[0], "cold point height", "km");
00145     NC_DEF_VAR("clp_p", NC_FLOAT, 3, &dims[0], "cold point pressure",
00146               "hPa");
00147     NC_DEF_VAR("clp_t", NC_FLOAT, 3, &dims[0], "cold point temperature",
00148               "K");
00149     if (h2o)
00150         NC_DEF_VAR("clp_q", NC_FLOAT, 3, &dims[0], "cold point water vapor",
00151                   "ppv");
00152     if (o3)
00153         NC_DEF_VAR("clp_o3", NC_FLOAT, 3, &dims[0], "cold point ozone",
00154                   "ppv");
00155
00156     NC_DEF_VAR("dyn_z", NC_FLOAT, 3, &dims[0],
00157               "dynamical tropopause height", "km");
00158     NC_DEF_VAR("dyn_p", NC_FLOAT, 3, &dims[0],
00159               "dynamical tropopause pressure", "hPa");
00160     NC_DEF_VAR("dyn_t", NC_FLOAT, 3, &dims[0],
00161               "dynamical tropopause temperature", "K");
00162     if (h2o)
00163         NC_DEF_VAR("dyn_q", NC_FLOAT, 3, &dims[0],
00164                   "dynamical tropopause water vapor", "ppv");
00165     if (o3)
00166         NC_DEF_VAR("dyn_o3", NC_FLOAT, 3, &dims[0],
00167                   "dynamical tropopause ozone", "ppv");
00168
00169     NC_DEF_VAR("wmo_1st_z", NC_FLOAT, 3, &dims[0],
00170               "WMO 1st tropopause height", "km");
00171     NC_DEF_VAR("wmo_1st_p", NC_FLOAT, 3, &dims[0],
00172               "WMO 1st tropopause pressure", "hPa");
00173     NC_DEF_VAR("wmo_1st_t", NC_FLOAT, 3, &dims[0],
00174               "WMO 1st tropopause temperature", "K");
00175     if (h2o)
00176         NC_DEF_VAR("wmo_1st_q", NC_FLOAT, 3, &dims[0],
00177                   "WMO 1st tropopause water vapor", "ppv");
00178     if (o3)
00179         NC_DEF_VAR("wmo_1st_o3", NC_FLOAT, 3, &dims[0],
00180                   "WMO 1st tropopause ozone", "ppv");
00181
00182     NC_DEF_VAR("wmo_2nd_z", NC_FLOAT, 3, &dims[0],
00183               "WMO 2nd tropopause height", "km");
00184     NC_DEF_VAR("wmo_2nd_p", NC_FLOAT, 3, &dims[0],
00185               "WMO 2nd tropopause pressure", "hPa");
00186     NC_DEF_VAR("wmo_2nd_t", NC_FLOAT, 3, &dims[0],
00187               "WMO 2nd tropopause temperature", "K");
00188     if (h2o)
00189         NC_DEF_VAR("wmo_2nd_q", NC_FLOAT, 3, &dims[0],
00190                   "WMO 2nd tropopause water vapor", "ppv");
00191     if (o3)
00192         NC_DEF_VAR("wmo_2nd_o3", NC_FLOAT, 3, &dims[0],
00193                   "WMO 2nd tropopause ozone", "ppv");

```

```

00194
00195     NC_DEF_VAR("ps", NC_FLOAT, 3, &dims[0], "surface pressure", "hPa");
00196     NC_DEF_VAR("zs", NC_FLOAT, 3, &dims[0], "surface height", "km");
00197
00198     /* End definition... */
00199     NC(nc_enddef(ncid));
00200
00201     /* Write longitude and latitude... */
00202     NC_PUT_DOUBLE("lat", lats, 0);
00203     NC_PUT_DOUBLE("lon", lons, 0);
00204 }
00205
00206 /* Write time... */
00207 start[0] = (size_t) nt;
00208 count[0] = 1;
00209 start[1] = 0;
00210 count[1] = (size_t) ny;
00211 start[2] = 0;
00212 count[2] = (size_t) nx;
00213 NC_PUT_DOUBLE("time", &met->time, 1);
00214
00215 /* Get cold point... */
00216 get_tropo(2, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00217          zs);
00218 NC_PUT_DOUBLE("clp_z", zt, 1);
00219 NC_PUT_DOUBLE("clp_p", pt, 1);
00220 NC_PUT_DOUBLE("clp_t", tt, 1);
00221 if (h2o)
00222     NC_PUT_DOUBLE("clp_q", qt, 1);
00223 if (o3)
00224     NC_PUT_DOUBLE("clp_o3", o3t, 1);
00225
00226 /* Get dynamical tropopause... */
00227 get_tropo(5, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00228          zs);
00229 NC_PUT_DOUBLE("dyn_z", zt, 1);
00230 NC_PUT_DOUBLE("dyn_p", pt, 1);
00231 NC_PUT_DOUBLE("dyn_t", tt, 1);
00232 if (h2o)
00233     NC_PUT_DOUBLE("dyn_q", qt, 1);
00234 if (o3)
00235     NC_PUT_DOUBLE("dyn_o3", o3t, 1);
00236
00237 /* Get WMO 1st tropopause... */
00238 get_tropo(3, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00239          zs);
00240 NC_PUT_DOUBLE("wmo_1st_z", zt, 1);
00241 NC_PUT_DOUBLE("wmo_1st_p", pt, 1);
00242 NC_PUT_DOUBLE("wmo_1st_t", tt, 1);
00243 if (h2o)
00244     NC_PUT_DOUBLE("wmo_1st_q", qt, 1);
00245 if (o3)
00246     NC_PUT_DOUBLE("wmo_1st_o3", o3t, 1);
00247
00248 /* Get WMO 2nd tropopause... */
00249 get_tropo(4, &ctl, clim, met, lons, nx, lats, ny, pt, zt, tt, qt, o3t, ps,
00250          zs);
00251 NC_PUT_DOUBLE("wmo_2nd_z", zt, 1);
00252 NC_PUT_DOUBLE("wmo_2nd_p", pt, 1);
00253 NC_PUT_DOUBLE("wmo_2nd_t", tt, 1);
00254 if (h2o)
00255     NC_PUT_DOUBLE("wmo_2nd_q", qt, 1);
00256 if (o3)
00257     NC_PUT_DOUBLE("wmo_2nd_o3", o3t, 1);
00258
00259 /* Write surface data... */
00260 NC_PUT_DOUBLE("ps", ps, 1);
00261 NC_PUT_DOUBLE("zs", zs, 1);
00262
00263 /* Increment time step counter... */
00264 nt++;
00265 }
00266
00267 /* Close file... */
00268 NC(nc_close(ncid));
00269
00270 /* Free... */
00271 free(clim);
00272 free(met);
00273
00274 return EXIT_SUCCESS;
00275 }
00276
00277 /*****
00278
00279 void get_tropo(
00280     int met_tropo,

```

```

00281     ctl_t * ctl,
00282     clim_t * clim,
00283     met_t * met,
00284     double *lons,
00285     int nx,
00286     double *lats,
00287     int ny,
00288     double *pt,
00289     double *zt,
00290     double *tt,
00291     double *qt,
00292     double *o3t,
00293     double *ps,
00294     double *zs) {
00295
00296     INTPOL_INIT;
00297
00298     ctl->met_tropo = met_tropo;
00299     read_met_tropo(ctl, clim, met);
00300 #pragma omp parallel for default(shared) private(ci,cw)
00301     for (int ix = 0; ix < nx; ix++)
00302         for (int iy = 0; iy < ny; iy++) {
00303             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00304                                &pt[iy * nx + ix], ci, cw, 1);
00305             intpol_met_space_2d(met, met->ps, lons[ix], lats[iy],
00306                                &ps[iy * nx + ix], ci, cw, 0);
00307             intpol_met_space_2d(met, met->zs, lons[ix], lats[iy],
00308                                &zs[iy * nx + ix], ci, cw, 0);
00309             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00310                                lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00311             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00312                                lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00313             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00314                                lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00315             intpol_met_space_3d(met, met->o3, pt[iy * nx + ix], lons[ix],
00316                                lats[iy], &o3t[iy * nx + ix], ci, cw, 0);
00317         }
00318 }

```

5.51 tropo_sample.c File Reference

Sample tropopause data set.

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

Macros

- `#define NT 744`
Maximum number of time steps.

Functions

- void `intpol_tropo_3d` (double time0, float array0[EX][EY], double time1, float array1[EX][EY], double lons[EX], double lats[EY], int nlon, int nlat, double time, double lon, double lat, int method, double *var, double *sigma)
3-D linear interpolation of tropopause data.
- int `main` (int argc, char *argv[])

5.51.1 Detailed Description

Sample tropopause data set.

Definition in file `tropo_sample.c`.

5.51.2 Macro Definition Documentation

5.51.2.1 NT `#define NT 744`

Maximum number of time steps.

Definition at line 32 of file [tropo_sample.c](#).

5.51.3 Function Documentation

5.51.3.1 `intpol_tropo_3d()` `void intpol_tropo_3d (`

```
double time0,
float array0[EX][EY],
double time1,
float array1[EX][EY],
double lons[EX],
double lats[EY],
int nlon,
int nlat,
double time,
double lon,
double lat,
int method,
double * var,
double * sigma )
```

3-D linear interpolation of tropopause data.

Definition at line 279 of file [tropo_sample.c](#).

```
00293     {
00294
00295     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00296
00297     int n = 0;
00298
00299     /* Adjust longitude... */
00300     if (lon < lons[0])
00301         lon += 360;
00302     else if (lon > lons[nlon - 1])
00303         lon -= 360;
00304
00305     /* Get indices... */
00306     int ix = locate_reg(lons, (int) nlon, lon);
00307     int iy = locate_reg(lats, (int) nlat, lat);
00308
00309     /* Calculate standard deviation... */
00310     *sigma = 0;
00311     for (int dx = 0; dx < 2; dx++)
00312         for (int dy = 0; dy < 2; dy++) {
00313             if (isfinite(array0[ix + dx][iy + dy])) {
00314                 mean += array0[ix + dx][iy + dy];
00315                 *sigma += SQR(array0[ix + dx][iy + dy]);
00316                 n++;
00317             }
00318             if (isfinite(array1[ix + dx][iy + dy])) {
00319                 mean += array1[ix + dx][iy + dy];
00320                 *sigma += SQR(array1[ix + dx][iy + dy]);
00321                 n++;
00322             }
00323         }
```

```

00324  if (n > 0)
00325      *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00326
00327  /* Linear interpolation... */
00328  if (method == 1 && isfinite(array0[ix][iy])
00329      && isfinite(array0[ix][iy + 1])
00330      && isfinite(array0[ix + 1][iy])
00331      && isfinite(array0[ix + 1][iy + 1])
00332      && isfinite(array1[ix][iy])
00333      && isfinite(array1[ix][iy + 1])
00334      && isfinite(array1[ix + 1][iy])
00335      && isfinite(array1[ix + 1][iy + 1])) {
00336
00337      aux00 = LIN(lons[ix], array0[ix][iy],
00338                  lons[ix + 1], array0[ix + 1][iy], lon);
00339      aux01 = LIN(lons[ix], array0[ix][iy + 1],
00340                  lons[ix + 1], array0[ix + 1][iy + 1], lon);
00341      aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00342
00343      aux10 = LIN(lons[ix], array1[ix][iy],
00344                  lons[ix + 1], array1[ix + 1][iy], lon);
00345      aux11 = LIN(lons[ix], array1[ix][iy + 1],
00346                  lons[ix + 1], array1[ix + 1][iy + 1], lon);
00347      aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00348
00349      *var = LIN(time0, aux0, time1, aux1, time);
00350  }
00351
00352  /* Nearest neighbor interpolation... */
00353  else {
00354      aux00 = NN(lons[ix], array0[ix][iy],
00355                 lons[ix + 1], array0[ix + 1][iy], lon);
00356      aux01 = NN(lons[ix], array0[ix][iy + 1],
00357                 lons[ix + 1], array0[ix + 1][iy + 1], lon);
00358      aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00359
00360      aux10 = NN(lons[ix], array1[ix][iy],
00361                 lons[ix + 1], array1[ix + 1][iy], lon);
00362      aux11 = NN(lons[ix], array1[ix][iy + 1],
00363                 lons[ix + 1], array1[ix + 1][iy + 1], lon);
00364      aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00365
00366      *var = NN(time0, aux0, time1, aux1, time);
00367  }
00368 }

```

Here is the call graph for this function:



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

Definition at line 59 of file [tropo_sample.c](#).

```

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, o30, o30sig;
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], tropo_t1[EX][EY],
00076     tropo_q0[EX][EY], tropo_q1[EX][EY], tropo_o30[EX][EY], tropo_o31[EX][EY];
00077
00078 static int ip, iq, it, it_old =
00079     -999, method, ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3,
00080     h2o, o3, ntime, nlon, nlat, ilon, ilat;
00081
00082 static size_t count[10], start[10];
00083
00084 /* Allocate... */
00085 ALLOC(atm, atm_t, 1);
00086
00087 /* Check arguments... */
00088 if (argc < 5)
00089     ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00090
00091 /* Read control parameters... */
00092 read_ctl(argv[1], argc, argv, &ctl);
00093 method =
00094     (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);
00095
00096 /* Read atmospheric data... */
00097 if (!read_atm(argv[5], &ctl, atm))
00098     ERRMSG("Cannot open file!");
00099
00100 /* Open tropopause file... */
00101 LOG(1, "Read tropopause data: %s", argv[3]);
00102 if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00103     ERRMSG("Cannot open file!");
00104
00105 /* Get dimensions... */
00106 NC_INQ_DIM("time", &ntime, 1, NT);
00107 NC_INQ_DIM("lat", &nlat, 1, EY);
00108 NC_INQ_DIM("lon", &nlon, 1, EX);
00109
00110 /* Read coordinates... */
00111 NC_GET_DOUBLE("time", times, 1);
00112 NC_GET_DOUBLE("lat", lats, 1);
00113 NC_GET_DOUBLE("lon", lons, 1);
00114
00115 /* Get variable indices... */
00116 sprintf(varname, "%s_z", argv[4]);
00117 NC(nc_inq_varid(ncid, varname, &varid_z));
00118 sprintf(varname, "%s_p", argv[4]);
00119 NC(nc_inq_varid(ncid, varname, &varid_p));
00120 sprintf(varname, "%s_t", argv[4]);
00121 NC(nc_inq_varid(ncid, varname, &varid_t));
00122 sprintf(varname, "%s_q", argv[4]);
00123 h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00124 sprintf(varname, "%s_o3", argv[4]);
00125 o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);
00126
00127 /* Set dimensions... */
00128 count[0] = 1;
00129 count[1] = (size_t) nlat;
00130 count[2] = (size_t) nlon;
00131
00132 /* Create file... */
00133 LOG(1, "Write tropopause sample data: %s", argv[2]);
00134 if (!(out = fopen(argv[2], "w")))
00135     ERRMSG("Cannot create file!");
00136
00137 /* Write header... */
00138 fprintf(out,
00139     "# $1 = time [s]\n"
00140     "# $2 = altitude [km]\n"
00141     "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00142 for (iq = 0; iq < ctl.nq; iq++)
00143     fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00144         ctl.qnt_unit[iq]);
00145 fprintf(out, "# $d = tropopause height (mean) [km]\n", 5 + ctl.nq);
00146 fprintf(out, "# $d = tropopause pressure (mean) [hPa]\n", 6 + ctl.nq);
00147 fprintf(out, "# $d = tropopause temperature (mean) [K]\n", 7 + ctl.nq);
00148 fprintf(out, "# $d = tropopause water vapor (mean) [ppv]\n", 8 + ctl.nq);
00149 fprintf(out, "# $d = tropopause ozone (mean) [ppv]\n", 9 + ctl.nq);
00150 fprintf(out, "# $d = tropopause height (sigma) [km]\n", 10 + ctl.nq);
00151 fprintf(out, "# $d = tropopause pressure (sigma) [hPa]\n", 11 + ctl.nq);
00152 fprintf(out, "# $d = tropopause temperature (sigma) [K]\n", 12 + ctl.nq);
00153 fprintf(out, "# $d = tropopause water vapor (sigma) [ppv]\n", 13 + ctl.nq);
00154 fprintf(out, "# $d = tropopause ozone (sigma) [ppv]\n", 14 + ctl.nq);
00155

```



```

00156  /* Loop over particles... */
00157  for (ip = 0; ip < atm->nnp; ip++) {
00158
00159      /* Check temporal ordering... */
00160      if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00161          ERRMSG("Time must be ascending!");
00162
00163      /* Check range... */
00164      if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00165          continue;
00166
00167      /* Read data... */
00168      it = locate_irr(times, (int) ntime, atm->time[ip]);
00169      if (it != it_old) {
00170
00171          time0 = times[it];
00172          start[0] = (size_t) it;
00173          NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00174          for (ilon = 0; ilon < nlon; ilon++)
00175              for (ilat = 0; ilat < nlat; ilat++)
00176                  tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00177          NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00178          for (ilon = 0; ilon < nlon; ilon++)
00179              for (ilat = 0; ilat < nlat; ilat++)
00180                  tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00181          NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00182          for (ilon = 0; ilon < nlon; ilon++)
00183              for (ilat = 0; ilat < nlat; ilat++)
00184                  tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00185          if (h2o) {
00186              NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00187              for (ilon = 0; ilon < nlon; ilon++)
00188                  for (ilat = 0; ilat < nlat; ilat++)
00189                      tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00190          } else
00191              for (ilon = 0; ilon < nlon; ilon++)
00192                  for (ilat = 0; ilat < nlat; ilat++)
00193                      tropo_q0[ilon][ilat] = GSL_NAN;
00194          if (o3) {
00195              NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
00196              for (ilon = 0; ilon < nlon; ilon++)
00197                  for (ilat = 0; ilat < nlat; ilat++)
00198                      tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
00199          } else
00200              for (ilon = 0; ilon < nlon; ilon++)
00201                  for (ilat = 0; ilat < nlat; ilat++)
00202                      tropo_o30[ilon][ilat] = GSL_NAN;
00203
00204          time1 = times[it + 1];
00205          start[0] = (size_t) it + 1;
00206          NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00207          for (ilon = 0; ilon < nlon; ilon++)
00208              for (ilat = 0; ilat < nlat; ilat++)
00209                  tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00210          NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00211          for (ilon = 0; ilon < nlon; ilon++)
00212              for (ilat = 0; ilat < nlat; ilat++)
00213                  tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00214          NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00215          for (ilon = 0; ilon < nlon; ilon++)
00216              for (ilat = 0; ilat < nlat; ilat++)
00217                  tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00218          if (h2o) {
00219              NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00220              for (ilon = 0; ilon < nlon; ilon++)
00221                  for (ilat = 0; ilat < nlat; ilat++)
00222                      tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00223          } else
00224              for (ilon = 0; ilon < nlon; ilon++)
00225                  for (ilat = 0; ilat < nlat; ilat++)
00226                      tropo_q1[ilon][ilat] = GSL_NAN;
00227          if (o3) {
00228              NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
00229              for (ilon = 0; ilon < nlon; ilon++)
00230                  for (ilat = 0; ilat < nlat; ilat++)
00231                      tropo_o31[ilon][ilat] = help[ilat * nlon + ilon];
00232          } else
00233              for (ilon = 0; ilon < nlon; ilon++)
00234                  for (ilat = 0; ilat < nlat; ilat++)
00235                      tropo_o31[ilon][ilat] = GSL_NAN;
00236      }
00237      it_old = it;
00238
00239      /* Interpolate... */
00240      intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00241                     lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00242                     atm->lat[ip], method, &z0, &z0sig);

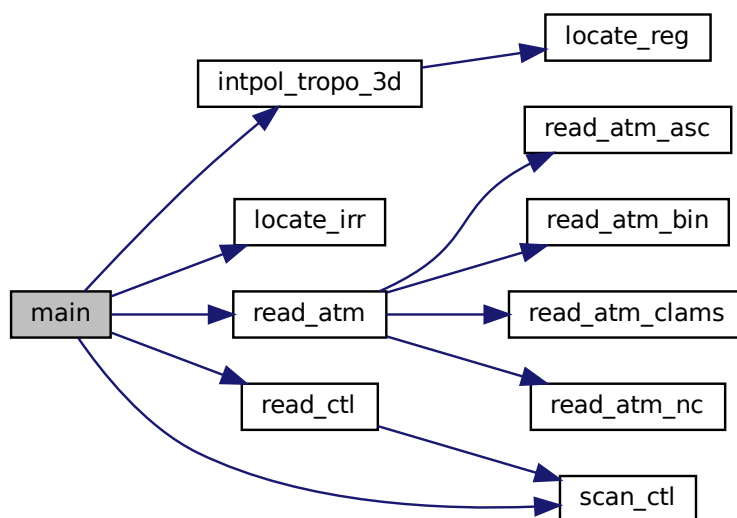
```

```

00243     intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00244                     lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00245                     atm->lat[ip], method, &p0, &p0sig);
00246     intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00247                     lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00248                     atm->lat[ip], method, &t0, &t0sig);
00249     intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00250                     lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00251                     atm->lat[ip], method, &q0, &q0sig);
00252     intpol_tropo_3d(time0, tropo_o30, time1, tropo_o31,
00253                     lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00254                     atm->lat[ip], method, &o30, &o30sig);
00255
00256     /* Write output... */
00257     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00258             atm->lon[ip], atm->lat[ip]);
00259     for (iq = 0; iq < ctl.nq; iq++) {
00260         fprintf(out, " ");
00261         fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00262     }
00263     fprintf(out, " %g %g %g %g %g %g %g %g %g %g\n",
00264             z0, p0, t0, q0, o30, z0sig, p0sig, t0sig, q0sig, o30sig);
00265 }
00266
00267 /* Close files... */
00268 fclose(out);
00269 NC(nc_close(ncid));
00270
00271 /* Free... */
00272 free(atm);
00273
00274 return EXIT_SUCCESS;
00275 }

```

Here is the call graph for this function:



5.52 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) 2013-2023 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 /* -----
00023 Dimensions...
00024 ----- */
00025
00026 #define NT 744
00027
00028 /* -----
00029 Functions...
00030 ----- */
00031
00032 void intpol_tropo_3d(
00033     double time0,
00034     float array0[EX][EY],
00035     double time1,
00036     float array1[EX][EY],
00037     double lons[EX],
00038     double lats[EY],
00039     int nlon,
00040     int nlat,
00041     double time,
00042     double lon,
00043     double lat,
00044     int method,
00045     double *var,
00046     double *sigma);
00047
00048 /* -----
00049 Main...
00050 ----- */
00051
00052 int main(
00053     int argc,
00054     char *argv[]) {
00055
00056     ctl_t ctl;
00057
00058     atm_t *atm;
00059
00060     static FILE *out;
00061
00062     static char varname[LEN];
00063
00064     static double times[NT], lons[EX], lats[EY], time0, time1, z0, z0sig,
00065         p0, p0sig, t0, t0sig, q0, q0sig, o30, o30sig;
00066
00067     static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00068         tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY], tropo_t1[EX][EY],
00069         tropo_q0[EX][EY], tropo_q1[EX][EY], tropo_o30[EX][EY], tropo_o31[EX][EY];
00070
00071     static int ip, iq, it, it_old =
00072         -999, method, ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3,
00073         h2o, o3, ntime, nlon, nlat, ilon, ilat;
00074
00075     static size_t count[10], start[10];
00076
00077     /* Allocate... */
00078     ALLOC(atm, atm_t, 1);
00079
00080     /* Check arguments... */
00081     if (argc < 5)
00082         ERRMSG("Give parameters: <ctl> <sample.tab> <tropo.nc> <var> <atm_in>");
00083
00084     /* Read control parameters... */
00085     read_ctl(argv[1], argc, argv, &ctl);
00086     method =
00087         (int) scan_ctl(argv[1], argc, argv, "TROPO_SAMPLE_METHOD", -1, "1", NULL);
00088
00089     /* Read atmospheric data... */
00090     if (!read_atm(argv[5], &ctl, atm))
00091         ERRMSG("Cannot open file!");
00092
00093     /* Open tropopause file... */
00094     LOG(1, "Read tropopause data: %s", argv[3]);
00095     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00096         ERRMSG("Cannot open file!");
00097
00098
00099
00100
00101
00102
00103

```

```

00104
00105 /* Get dimensions... */
00106 NC_INQ_DIM("time", &ntime, 1, NT);
00107 NC_INQ_DIM("lat", &nlat, 1, EY);
00108 NC_INQ_DIM("lon", &nlon, 1, EX);
00109
00110 /* Read coordinates... */
00111 NC_GET_DOUBLE("time", times, 1);
00112 NC_GET_DOUBLE("lat", lats, 1);
00113 NC_GET_DOUBLE("lon", lons, 1);
00114
00115 /* Get variable indices... */
00116 sprintf(varname, "%s_z", argv[4]);
00117 NC(nc_inq_varid(ncid, varname, &varid_z));
00118 sprintf(varname, "%s_p", argv[4]);
00119 NC(nc_inq_varid(ncid, varname, &varid_p));
00120 sprintf(varname, "%s_t", argv[4]);
00121 NC(nc_inq_varid(ncid, varname, &varid_t));
00122 sprintf(varname, "%s_q", argv[4]);
00123 h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00124 sprintf(varname, "%s_o3", argv[4]);
00125 o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);
00126
00127 /* Set dimensions... */
00128 count[0] = 1;
00129 count[1] = (size_t) nlat;
00130 count[2] = (size_t) nlon;
00131
00132 /* Create file... */
00133 LOG(1, "Write tropopause sample data: %s", argv[2]);
00134 if (!(out = fopen(argv[2], "w")))
00135     ERRMSG("Cannot create file!");
00136
00137 /* Write header... */
00138 fprintf(out,
00139         "# $1 = time [s]\n"
00140         "# $2 = altitude [km]\n"
00141         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00142 for (iq = 0; iq < ctl.nq; iq++)
00143     fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00144             ctl.qnt_unit[iq]);
00145 fprintf(out, "# $d = tropopause height (mean) [km]\n", 5 + ctl.nq);
00146 fprintf(out, "# $d = tropopause pressure (mean) [hPa]\n", 6 + ctl.nq);
00147 fprintf(out, "# $d = tropopause temperature (mean) [K]\n", 7 + ctl.nq);
00148 fprintf(out, "# $d = tropopause water vapor (mean) [ppv]\n", 8 + ctl.nq);
00149 fprintf(out, "# $d = tropopause ozone (mean) [ppv]\n", 9 + ctl.nq);
00150 fprintf(out, "# $d = tropopause height (sigma) [km]\n", 10 + ctl.nq);
00151 fprintf(out, "# $d = tropopause pressure (sigma) [hPa]\n", 11 + ctl.nq);
00152 fprintf(out, "# $d = tropopause temperature (sigma) [K]\n", 12 + ctl.nq);
00153 fprintf(out, "# $d = tropopause water vapor (sigma) [ppv]\n", 13 + ctl.nq);
00154 fprintf(out, "# $d = tropopause ozone (sigma) [ppv]\n", 14 + ctl.nq);
00155
00156 /* Loop over particles... */
00157 for (ip = 0; ip < atm->nnp; ip++) {
00158
00159     /* Check temporal ordering... */
00160     if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00161         ERRMSG("Time must be ascending!");
00162
00163     /* Check range... */
00164     if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00165         continue;
00166
00167     /* Read data... */
00168     it = locate_irr(times, (int) ntime, atm->time[ip]);
00169     if (it != it_old) {
00170
00171         time0 = times[it];
00172         start[0] = (size_t) it;
00173         NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00174         for (ilon = 0; ilon < nlon; ilon++)
00175             for (ilat = 0; ilat < nlat; ilat++)
00176                 tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00177         NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00178         for (ilon = 0; ilon < nlon; ilon++)
00179             for (ilat = 0; ilat < nlat; ilat++)
00180                 tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00181         NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00182         for (ilon = 0; ilon < nlon; ilon++)
00183             for (ilat = 0; ilat < nlat; ilat++)
00184                 tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00185         if (h2o) {
00186             NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00187             for (ilon = 0; ilon < nlon; ilon++)
00188                 for (ilat = 0; ilat < nlat; ilat++)
00189                     tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00190         } else

```

```

00191         for (ilon = 0; ilon < nlon; ilon++)
00192             for (ilat = 0; ilat < nlat; ilat++)
00193                 tropo_q0[ilon][ilat] = GSL_NAN;
00194     if (o3) {
00195         NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
00196         for (ilon = 0; ilon < nlon; ilon++)
00197             for (ilat = 0; ilat < nlat; ilat++)
00198                 tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
00199     } else
00200         for (ilon = 0; ilon < nlon; ilon++)
00201             for (ilat = 0; ilat < nlat; ilat++)
00202                 tropo_o30[ilon][ilat] = GSL_NAN;
00203
00204     time1 = times[it + 1];
00205     start[0] = (size_t) it + 1;
00206     NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00207     for (ilon = 0; ilon < nlon; ilon++)
00208         for (ilat = 0; ilat < nlat; ilat++)
00209             tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00210     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00211     for (ilon = 0; ilon < nlon; ilon++)
00212         for (ilat = 0; ilat < nlat; ilat++)
00213             tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00214     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00215     for (ilon = 0; ilon < nlon; ilon++)
00216         for (ilat = 0; ilat < nlat; ilat++)
00217             tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00218     if (h2o) {
00219         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00220         for (ilon = 0; ilon < nlon; ilon++)
00221             for (ilat = 0; ilat < nlat; ilat++)
00222                 tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00223     } else
00224         for (ilon = 0; ilon < nlon; ilon++)
00225             for (ilat = 0; ilat < nlat; ilat++)
00226                 tropo_q1[ilon][ilat] = GSL_NAN;;
00227     if (o3) {
00228         NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
00229         for (ilon = 0; ilon < nlon; ilon++)
00230             for (ilat = 0; ilat < nlat; ilat++)
00231                 tropo_o31[ilon][ilat] = help[ilat * nlon + ilon];
00232     } else
00233         for (ilon = 0; ilon < nlon; ilon++)
00234             for (ilat = 0; ilat < nlat; ilat++)
00235                 tropo_o31[ilon][ilat] = GSL_NAN;;
00236 }
00237 it_old = it;
00238
00239 /* Interpolate... */
00240 intpol_tropo_3d(time0, tropo_z0, time1, tropo_z1,
00241                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00242                atm->lat[ip], method, &z0, &z0sig);
00243 intpol_tropo_3d(time0, tropo_p0, time1, tropo_p1,
00244                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00245                atm->lat[ip], method, &p0, &p0sig);
00246 intpol_tropo_3d(time0, tropo_t0, time1, tropo_t1,
00247                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00248                atm->lat[ip], method, &t0, &t0sig);
00249 intpol_tropo_3d(time0, tropo_q0, time1, tropo_q1,
00250                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00251                atm->lat[ip], method, &q0, &q0sig);
00252 intpol_tropo_3d(time0, tropo_o30, time1, tropo_o31,
00253                lons, lats, nlon, nlat, atm->time[ip], atm->lon[ip],
00254                atm->lat[ip], method, &o30, &o30sig);
00255
00256 /* Write output... */
00257 fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00258         atm->lon[ip], atm->lat[ip]);
00259 for (iq = 0; iq < ctl.nq; iq++) {
00260     fprintf(out, " ");
00261     fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00262 }
00263 fprintf(out, " %g %g %g %g %g %g %g %g %g %g\n",
00264         z0, p0, t0, q0, o30, z0sig, p0sig, t0sig, q0sig, o30sig);
00265 }
00266
00267 /* Close files... */
00268 fclose(out);
00269 NC(nc_close(ncid));
00270
00271 /* Free... */
00272 free(atm);
00273
00274 return EXIT_SUCCESS;
00275 }
00276
00277 /*****

```

```

00278
00279 void intpol_tropo_3d(
00280     double time0,
00281     float array0[EX][EY],
00282     double time1,
00283     float array1[EX][EY],
00284     double lons[EX],
00285     double lats[EY],
00286     int nlon,
00287     int nlat,
00288     double time,
00289     double lon,
00290     double lat,
00291     int method,
00292     double *var,
00293     double *sigma) {
00294
00295     double aux0, aux1, aux00, aux01, aux10, aux11, mean = 0;
00296
00297     int n = 0;
00298
00299     /* Adjust longitude... */
00300     if (lon < lons[0])
00301         lon += 360;
00302     else if (lon > lons[nlon - 1])
00303         lon -= 360;
00304
00305     /* Get indices... */
00306     int ix = locate_reg(lons, (int) nlon, lon);
00307     int iy = locate_reg(lats, (int) nlat, lat);
00308
00309     /* Calculate standard deviation... */
00310     *sigma = 0;
00311     for (int dx = 0; dx < 2; dx++)
00312         for (int dy = 0; dy < 2; dy++) {
00313             if (isfinite(array0[ix + dx][iy + dy])) {
00314                 mean += array0[ix + dx][iy + dy];
00315                 *sigma += SQR(array0[ix + dx][iy + dy]);
00316                 n++;
00317             }
00318             if (isfinite(array1[ix + dx][iy + dy])) {
00319                 mean += array1[ix + dx][iy + dy];
00320                 *sigma += SQR(array1[ix + dx][iy + dy]);
00321                 n++;
00322             }
00323         }
00324     if (n > 0)
00325         *sigma = sqrt(GSL_MAX(*sigma / n - SQR(mean / n), 0.0));
00326
00327     /* Linear interpolation... */
00328     if (method == 1 && isfinite(array0[ix][iy])
00329         && isfinite(array0[ix][iy + 1])
00330         && isfinite(array0[ix + 1][iy])
00331         && isfinite(array0[ix + 1][iy + 1])
00332         && isfinite(array1[ix][iy])
00333         && isfinite(array1[ix][iy + 1])
00334         && isfinite(array1[ix + 1][iy])
00335         && isfinite(array1[ix + 1][iy + 1])) {
00336
00337         aux00 = LIN(lons[ix], array0[ix][iy],
00338             lons[ix + 1], array0[ix + 1][iy], lon);
00339         aux01 = LIN(lons[ix], array0[ix][iy + 1],
00340             lons[ix + 1], array0[ix + 1][iy + 1], lon);
00341         aux0 = LIN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00342
00343         aux10 = LIN(lons[ix], array1[ix][iy],
00344             lons[ix + 1], array1[ix + 1][iy], lon);
00345         aux11 = LIN(lons[ix], array1[ix][iy + 1],
00346             lons[ix + 1], array1[ix + 1][iy + 1], lon);
00347         aux1 = LIN(lats[iy], aux10, lats[iy + 1], aux11, lat);
00348
00349         *var = LIN(time0, aux0, time1, aux1, time);
00350     }
00351
00352     /* Nearest neighbor interpolation... */
00353     else {
00354         aux00 = NN(lons[ix], array0[ix][iy],
00355             lons[ix + 1], array0[ix + 1][iy], lon);
00356         aux01 = NN(lons[ix], array0[ix][iy + 1],
00357             lons[ix + 1], array0[ix + 1][iy + 1], lon);
00358         aux0 = NN(lats[iy], aux00, lats[iy + 1], aux01, lat);
00359
00360         aux10 = NN(lons[ix], array1[ix][iy],
00361             lons[ix + 1], array1[ix + 1][iy], lon);
00362         aux11 = NN(lons[ix], array1[ix][iy + 1],
00363             lons[ix + 1], array1[ix + 1][iy + 1], lon);
00364         aux1 = NN(lats[iy], aux10, lats[iy + 1], aux11, lat);

```

```
00365
00366     *var = NN(time0, aux0, time1, aux1, time);
00367   }
00368 }
```

5.53 tropo_zm.c File Reference

Extract zonal mean of tropopause data set.

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

Macros

- `#define NT 744`
Maximum number of time steps.

Functions

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

5.53.1 Detailed Description

Extract zonal mean of tropopause data set.

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

5.53.2 Macro Definition Documentation

5.53.2.1 NT `#define NT 744`

Maximum number of time steps.

Definition at line 32 of file [tropo_zm.c](#).

5.53.3 Function Documentation

```

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

```

Definition at line 38 of file [tropo_zm.c](#).

```

00040     {
00041
00042     ctl_t ctl;
00043
00044     static FILE *out;
00045
00046     static char tstr[LEN], varname[LEN];
00047
00048     static double time0, lons[EX], lats[EY], zm[EY], zs[EY], pm[EY],
00049         ps[EY], tm[EY], ts[EY], qm[EY], qs[EY], o3m[EY], o3s[EY];
00050
00051     static float help[EX * EY], tropo_z0[EX][EY], tropo_p0[EX][EY],
00052         tropo_t0[EX][EY], tropo_q0[EX][EY], tropo_o30[EX][EY];
00053
00054     static int ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3, h2o,
00055         o3, n[EY], nt[EY], year, mon, day, init, ntime, nlon, nlat, ilon, ilat;
00056
00057     static size_t count[10], start[10];
00058
00059     /* Check arguments... */
00060     if (argc < 5)
00061         ERRMSG("Give parameters: <ctl> <zm.tab> <var> <tropo.nc>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065
00066     /* Loop over tropopause files... */
00067     for (int iarg = 4; iarg < argc; iarg++) {
00068
00069         /* Open tropopause file... */
00070         LOG(1, "Read tropopause data: %s", argv[iarg]);
00071         if (nc_open(argv[iarg], NC_NOWRITE, &ncid) != NC_NOERR)
00072             ERRMSG("Cannot open file!");
00073
00074         /* Get dimensions... */
00075         NC_INQ_DIM("time", &ntime, 1, NT);
00076         NC_INQ_DIM("lat", &nlat, 1, EY);
00077         NC_INQ_DIM("lon", &nlon, 1, EX);
00078
00079         /* Read coordinates... */
00080         NC_GET_DOUBLE("lat", lats, 1);
00081         NC_GET_DOUBLE("lon", lons, 1);
00082
00083         /* Get variable indices... */
00084         sprintf(varname, "%s_z", argv[3]);
00085         NC(nc_inq_varid(ncid, varname, &varid_z));
00086         sprintf(varname, "%s_p", argv[3]);
00087         NC(nc_inq_varid(ncid, varname, &varid_p));
00088         sprintf(varname, "%s_t", argv[3]);
00089         NC(nc_inq_varid(ncid, varname, &varid_t));
00090         sprintf(varname, "%s_q", argv[3]);
00091         h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00092         sprintf(varname, "%s_o3", argv[3]);
00093         o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);
00094
00095         /* Set dimensions... */
00096         count[0] = 1;
00097         count[1] = (size_t) nlat;
00098         count[2] = (size_t) nlon;
00099
00100         /* Loop over time steps... */
00101         for (int it = 0; it < ntime; it++) {
00102
00103             /* Get time from filename... */
00104             if (!init) {
00105                 init = 1;
00106                 size_t len = strlen(argv[iarg]);
00107                 sprintf(tstr, "%.4s", &argv[iarg][len - 13]);
00108                 year = atoi(tstr);
00109                 sprintf(tstr, "%.2s", &argv[iarg][len - 8]);
00110                 mon = atoi(tstr);
00111                 sprintf(tstr, "%.2s", &argv[iarg][len - 5]);
00112                 day = atoi(tstr);
00113                 time2jsec(year, mon, day, 0, 0, 0, 0, &time0);
00114             }
00115
00116             /* Read data... */
00117             start[0] = (size_t) it;
00118             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00119             for (ilon = 0; ilon < nlon; ilon++)

```



```

00120         for (ilat = 0; ilat < nlat; ilat++)
00121             tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00122     NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00123     for (ilon = 0; ilon < nlon; ilon++)
00124         for (ilat = 0; ilat < nlat; ilat++)
00125             tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00126     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00127     for (ilon = 0; ilon < nlon; ilon++)
00128         for (ilat = 0; ilat < nlat; ilat++)
00129             tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00130     if (h2o) {
00131         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00132         for (ilon = 0; ilon < nlon; ilon++)
00133             for (ilat = 0; ilat < nlat; ilat++)
00134                 tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00135     } else
00136         for (ilon = 0; ilon < nlon; ilon++)
00137             for (ilat = 0; ilat < nlat; ilat++)
00138                 tropo_q0[ilon][ilat] = GSL_NAN;
00139     if (o3) {
00140         NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
00141         for (ilon = 0; ilon < nlon; ilon++)
00142             for (ilat = 0; ilat < nlat; ilat++)
00143                 tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
00144     } else
00145         for (ilon = 0; ilon < nlon; ilon++)
00146             for (ilat = 0; ilat < nlat; ilat++)
00147                 tropo_o30[ilon][ilat] = GSL_NAN;
00148
00149     /* Averaging... */
00150     for (ilat = 0; ilat < nlat; ilat++)
00151         for (ilon = 0; ilon < nlon; ilon++) {
00152             nt[ilat]++;
00153             if (isfinite(tropo_z0[ilon][ilat])
00154                 && isfinite(tropo_p0[ilon][ilat])
00155                 && isfinite(tropo_t0[ilon][ilat])
00156                 && (!h2o || isfinite(tropo_q0[ilon][ilat]))
00157                 && (!o3 || isfinite(tropo_o30[ilon][ilat]))) {
00158                 zm[ilat] += tropo_z0[ilon][ilat];
00159                 zs[ilat] += SQR(tropo_z0[ilon][ilat]);
00160                 pm[ilat] += tropo_p0[ilon][ilat];
00161                 ps[ilat] += SQR(tropo_p0[ilon][ilat]);
00162                 tm[ilat] += tropo_t0[ilon][ilat];
00163                 ts[ilat] += SQR(tropo_t0[ilon][ilat]);
00164                 qm[ilat] += tropo_q0[ilon][ilat];
00165                 qs[ilat] += SQR(tropo_q0[ilon][ilat]);
00166                 o3m[ilat] += tropo_o30[ilon][ilat];
00167                 o3s[ilat] += SQR(tropo_o30[ilon][ilat]);
00168                 n[ilat]++;
00169             }
00170         }
00171     }
00172
00173     /* Close files... */
00174     NC(nc_close(ncid));
00175 }
00176
00177 /* Normalize... */
00178 for (ilat = 0; ilat < nlat; ilat++)
00179     if (n[ilat] > 0) {
00180         zm[ilat] /= n[ilat];
00181         pm[ilat] /= n[ilat];
00182         tm[ilat] /= n[ilat];
00183         qm[ilat] /= n[ilat];
00184         o3m[ilat] /= n[ilat];
00185         double aux = zs[ilat] / n[ilat] - SQR(zm[ilat]);
00186         zs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00187         aux = ps[ilat] / n[ilat] - SQR(pm[ilat]);
00188         ps[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00189         aux = ts[ilat] / n[ilat] - SQR(tm[ilat]);
00190         ts[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00191         aux = qs[ilat] / n[ilat] - SQR(qm[ilat]);
00192         qs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00193         aux = o3s[ilat] / n[ilat] - SQR(o3m[ilat]);
00194         o3s[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00195     }
00196
00197 /* Create file... */
00198 LOG(1, "Write tropopause zonal mean data: %s", argv[2]);
00199 if (!(out = fopen(argv[2], "w")))
00200     ERRMSG("Cannot create file!");
00201
00202 /* Write header... */
00203 fprintf(out,
00204     "# $1 = time [s]\n"
00205     "# $2 = latitude [deg]\n"
00206     "# $3 = tropopause height (mean) [km]\n"

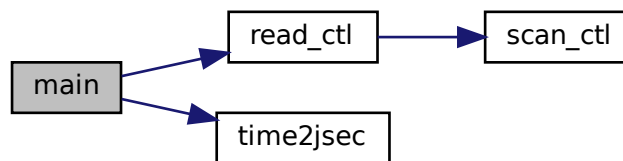
```

```

00207     "# $4 = tropopause pressure (mean) [hPa]\n"
00208     "# $5 = tropopause temperature (mean) [K]\n"
00209     "# $6 = tropopause water vapor (mean) [ppv]\n"
00210     "# $7 = tropopause ozone (mean) [ppv]\n"
00211     "# $8 = tropopause height (sigma) [km]\n"
00212     "# $9 = tropopause pressure (sigma) [hPa]\n"
00213     "# $10 = tropopause temperature (sigma) [K]\n"
00214     "# $11 = tropopause water vapor (sigma) [ppv]\n"
00215     "# $12 = tropopause ozone (sigma) [ppv]\n"
00216     "# $13 = number of data points\n"
00217     "# $14 = occurrence frequency [%]\n\n");
00218
00219 /* Write output... */
00220 for (ilat = 0; ilat < nlat; ilat++)
00221     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %d %g\n",
00222         time0, lats[ilat], zm[ilat], pm[ilat], tm[ilat], qm[ilat],
00223         o3m[ilat], zs[ilat], ps[ilat], ts[ilat], qs[ilat], o3s[ilat],
00224         n[ilat], 100. * n[ilat] / nt[ilat]);
00225
00226 /* Close files... */
00227 fclose(out);
00228
00229 return EXIT_SUCCESS;
00230 }

```

Here is the call graph for this function:



5.54 tropo_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) 2013-2023 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NT 744
00033
00034 /* -----
00035  Main...
00036  ----- */
00037
00038 int main(
00039     int argc,
00040     char *argv[]) {

```

```

00041
00042     ctl_t ctl;
00043
00044     static FILE *out;
00045
00046     static char tstr[LEN], varname[LEN];
00047
00048     static double time0, lons[EX], lats[EY], zm[EY], zs[EY], pm[EY],
00049         ps[EY], tm[EY], ts[EY], qm[EY], qs[EY], o3m[EY], o3s[EY];
00050
00051     static float help[EX * EY], tropo_z0[EX][EY], tropo_p0[EX][EY],
00052         tropo_t0[EX][EY], tropo_q0[EX][EY], tropo_o30[EX][EY];
00053
00054     static int ncid, varid, varid_z, varid_p, varid_t, varid_q, varid_o3, h2o,
00055         o3, n[EY], nt[EY], year, mon, day, init, ntime, nlon, nlat, ilon, ilat;
00056
00057     static size_t count[10], start[10];
00058
00059     /* Check arguments... */
00060     if (argc < 5)
00061         ERRMSG("Give parameters: <ctl> <zm.tab> <var> <tropo.nc>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065
00066     /* Loop over tropopause files... */
00067     for (int iarg = 4; iarg < argc; iarg++) {
00068
00069         /* Open tropopause file... */
00070         LOG(1, "Read tropopause data: %s", argv[iarg]);
00071         if (nc_open(argv[iarg], NC_NOWRITE, &ncid) != NC_NOERR)
00072             ERRMSG("Cannot open file!");
00073
00074         /* Get dimensions... */
00075         NC_INQ_DIM("time", &ntime, 1, NT);
00076         NC_INQ_DIM("lat", &nlat, 1, EY);
00077         NC_INQ_DIM("lon", &nlon, 1, EX);
00078
00079         /* Read coordinates... */
00080         NC_GET_DOUBLE("lat", lats, 1);
00081         NC_GET_DOUBLE("lon", lons, 1);
00082
00083         /* Get variable indices... */
00084         sprintf(varname, "%s_z", argv[3]);
00085         NC(nc_inq_varid(ncid, varname, &varid_z));
00086         sprintf(varname, "%s_p", argv[3]);
00087         NC(nc_inq_varid(ncid, varname, &varid_p));
00088         sprintf(varname, "%s_t", argv[3]);
00089         NC(nc_inq_varid(ncid, varname, &varid_t));
00090         sprintf(varname, "%s_q", argv[3]);
00091         h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00092         sprintf(varname, "%s_o3", argv[3]);
00093         o3 = (nc_inq_varid(ncid, varname, &varid_o3) == NC_NOERR);
00094
00095         /* Set dimensions... */
00096         count[0] = 1;
00097         count[1] = (size_t) nlat;
00098         count[2] = (size_t) nlon;
00099
00100         /* Loop over time steps... */
00101         for (int it = 0; it < ntime; it++) {
00102
00103             /* Get time from filename... */
00104             if (!init) {
00105                 init = 1;
00106                 size_t len = strlen(argv[iarg]);
00107                 sprintf(tstr, "%.4s", &argv[iarg][len - 13]);
00108                 year = atoi(tstr);
00109                 sprintf(tstr, "%.2s", &argv[iarg][len - 8]);
00110                 mon = atoi(tstr);
00111                 sprintf(tstr, "%.2s", &argv[iarg][len - 5]);
00112                 day = atoi(tstr);
00113                 time2jsec(year, mon, day, 0, 0, 0, 0, &time0);
00114             }
00115
00116             /* Read data... */
00117             start[0] = (size_t) it;
00118             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00119             for (ilon = 0; ilon < nlon; ilon++)
00120                 for (ilat = 0; ilat < nlat; ilat++)
00121                     tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00122             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00123             for (ilon = 0; ilon < nlon; ilon++)
00124                 for (ilat = 0; ilat < nlat; ilat++)
00125                     tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00126             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00127             for (ilon = 0; ilon < nlon; ilon++)

```

```

00128         for (ilat = 0; ilat < nlat; ilat++)
00129             tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00130     if (h2o) {
00131         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00132         for (ilon = 0; ilon < nlon; ilon++)
00133             for (ilat = 0; ilat < nlat; ilat++)
00134                 tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00135     } else
00136         for (ilon = 0; ilon < nlon; ilon++)
00137             for (ilat = 0; ilat < nlat; ilat++)
00138                 tropo_q0[ilon][ilat] = GSL_NAN;
00139     if (o3) {
00140         NC(nc_get_vara_float(ncid, varid_o3, start, count, help));
00141         for (ilon = 0; ilon < nlon; ilon++)
00142             for (ilat = 0; ilat < nlat; ilat++)
00143                 tropo_o30[ilon][ilat] = help[ilat * nlon + ilon];
00144     } else
00145         for (ilon = 0; ilon < nlon; ilon++)
00146             for (ilat = 0; ilat < nlat; ilat++)
00147                 tropo_o30[ilon][ilat] = GSL_NAN;
00148
00149     /* Averaging... */
00150     for (ilat = 0; ilat < nlat; ilat++)
00151         for (ilon = 0; ilon < nlon; ilon++) {
00152             nt[ilat]++;
00153             if (isfinite(tropo_z0[ilon][ilat])
00154                 && isfinite(tropo_p0[ilon][ilat])
00155                 && isfinite(tropo_t0[ilon][ilat])
00156                 && (!h2o || isfinite(tropo_q0[ilon][ilat]))
00157                 && (!o3 || isfinite(tropo_o30[ilon][ilat]))) {
00158                 zm[ilat] += tropo_z0[ilon][ilat];
00159                 zs[ilat] += SQR(tropo_z0[ilon][ilat]);
00160                 pm[ilat] += tropo_p0[ilon][ilat];
00161                 ps[ilat] += SQR(tropo_p0[ilon][ilat]);
00162                 tm[ilat] += tropo_t0[ilon][ilat];
00163                 ts[ilat] += SQR(tropo_t0[ilon][ilat]);
00164                 qm[ilat] += tropo_q0[ilon][ilat];
00165                 qs[ilat] += SQR(tropo_q0[ilon][ilat]);
00166                 o3m[ilat] += tropo_o30[ilon][ilat];
00167                 o3s[ilat] += SQR(tropo_o30[ilon][ilat]);
00168                 n[ilat]++;
00169             }
00170         }
00171     }
00172
00173     /* Close files... */
00174     NC(nc_close(ncid));
00175 }
00176
00177 /* Normalize... */
00178 for (ilat = 0; ilat < nlat; ilat++)
00179     if (n[ilat] > 0) {
00180         zm[ilat] /= n[ilat];
00181         pm[ilat] /= n[ilat];
00182         tm[ilat] /= n[ilat];
00183         qm[ilat] /= n[ilat];
00184         o3m[ilat] /= n[ilat];
00185         double aux = zs[ilat] / n[ilat] - SQR(zm[ilat]);
00186         zs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00187         aux = ps[ilat] / n[ilat] - SQR(pm[ilat]);
00188         ps[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00189         aux = ts[ilat] / n[ilat] - SQR(tm[ilat]);
00190         ts[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00191         aux = qs[ilat] / n[ilat] - SQR(qm[ilat]);
00192         qs[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00193         aux = o3s[ilat] / n[ilat] - SQR(o3m[ilat]);
00194         o3s[ilat] = aux > 0 ? sqrt(aux) : 0.0;
00195     }
00196
00197 /* Create file... */
00198 LOG(1, "Write tropopause zonal mean data: %s", argv[2]);
00199 if (!(out = fopen(argv[2], "w")))
00200     ERRMSG("Cannot create file!");
00201
00202 /* Write header... */
00203 fprintf(out,
00204     "# $1 = time [s]\n"
00205     "# $2 = latitude [deg]\n"
00206     "# $3 = tropopause height (mean) [km]\n"
00207     "# $4 = tropopause pressure (mean) [hPa]\n"
00208     "# $5 = tropopause temperature (mean) [K]\n"
00209     "# $6 = tropopause water vapor (mean) [ppv]\n"
00210     "# $7 = tropopause ozone (mean) [ppv]\n"
00211     "# $8 = tropopause height (sigma) [km]\n"
00212     "# $9 = tropopause pressure (sigma) [hPa]\n"
00213     "# $10 = tropopause temperature (sigma) [K]\n"
00214     "# $11 = tropopause water vapor (sigma) [ppv]\n"

```

```

00215     "# $12 = tropopause ozone (sigma) [ppv]\n"
00216     "# $13 = number of data points\n"
00217     "# $14 = occurrence frequency [%%]\n\n");
00218
00219     /* Write output... */
00220     for (ilat = 0; ilat < nlat; ilat++)
00221         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %d %g\n",
00222             time0, lats[ilat], zm[ilat], pm[ilat], tm[ilat], qm[ilat],
00223             o3m[ilat], zs[ilat], ps[ilat], ts[ilat], qs[ilat], o3s[ilat],
00224             n[ilat], 100. * n[ilat] / nt[ilat]);
00225
00226     /* Close files... */
00227     fclose(out);
00228
00229     return EXIT_SUCCESS;
00230 }

```

5.55 wind.c File Reference

Create meteorological data files with synthetic wind fields.

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

Functions

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

5.55.1 Detailed Description

Create meteorological data files with synthetic wind fields.

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

5.55.2 Function Documentation

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

Definition at line 31 of file [wind.c](#).

```

00033     {
00034
00035     ctl_t ctl;
00036
00037     static char filename[LEN];
00038
00039     static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00040         u0, u1, w0, alpha;
00041
00042     static float *dataT, *dataU, *dataV, *dataW;
00043
00044     static int ncid, varid, dims[4], idx, ix, iy, iz, nx, ny, nz,
00045         year, mon, day, hour, min, sec;
00046
00047     static size_t start[4], count[4];
00048
00049     /* Allocate... */
00050     ALLOC(dataT, float,
00051         EP * EY * EX);

```

```

00052  ALLOC(dataU, float,
00053          EP * EY * EX);
00054  ALLOC(dataV, float,
00055          EP * EY * EX);
00056  ALLOC(dataW, float,
00057          EP * EY * EX);
00058
00059  /* Check arguments... */
00060  if (argc < 3)
00061      ERRMSG("Give parameters: <ctl> <metbase>");
00062
00063  /* Read control parameters... */
00064  read_ctl(argv[1], argc, argv, &ctl);
00065  t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00066  nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00067  ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00068  nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00069  z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00070  z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00071  u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00072  u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00073  w0 = scan_ctl(argv[1], argc, argv, "WIND_W0", -1, "0", NULL);
00074  alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00075
00076  /* Check dimensions... */
00077  if (nx < 1 || nx > EX)
00078      ERRMSG("Set 1 <= NX <= MAX!");
00079  if (ny < 1 || ny > EY)
00080      ERRMSG("Set 1 <= NY <= MAX!");
00081  if (nz < 1 || nz > EP)
00082      ERRMSG("Set 1 <= NZ <= MAX!");
00083
00084  /* Get time... */
00085  jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00086  t0 = year * 10000. + mon * 100. + day + hour / 24.;
00087
00088  /* Set filename... */
00089  sprintf(filename, "%s_%d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00090
00091  /* Create netCDF file... */
00092  NC(nc_create(filename, NC_CLOBBER, &ncid));
00093
00094  /* Create dimensions... */
00095  NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00096  NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00097  NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00098  NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00099
00100  /* Create variables... */
00101  NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time", "day as %Y%m%d.%f");
00102  NC_DEF_VAR("lev", NC_DOUBLE, 1, &dims[1], "air_pressure", "Pa");
00103  NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[2], "latitude", "degrees_north");
00104  NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[3], "longitude", "degrees_east");
00105  NC_DEF_VAR("T", NC_FLOAT, 4, &dims[0], "Temperature", "K");
00106  NC_DEF_VAR("U", NC_FLOAT, 4, &dims[0], "zonal wind", "m s**-1");
00107  NC_DEF_VAR("V", NC_FLOAT, 4, &dims[0], "meridional wind", "m s**-1");
00108  NC_DEF_VAR("W", NC_FLOAT, 4, &dims[0], "vertical velocity", "Pa s**-1");
00109
00110  /* End definition... */
00111  NC(nc_enddef(ncid));
00112
00113  /* Set coordinates... */
00114  for (ix = 0; ix < nx; ix++)
00115      dataLon[ix] = 360.0 / nx * (double) ix;
00116  for (iy = 0; iy < ny; iy++)
00117      dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00118  for (iz = 0; iz < nz; iz++)
00119      dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00120
00121  /* Write coordinates... */
00122  NC_PUT_DOUBLE("time", &t0, 0);
00123  NC_PUT_DOUBLE("lev", dataZ, 0);
00124  NC_PUT_DOUBLE("lat", dataLat, 0);
00125  NC_PUT_DOUBLE("lon", dataLon, 0);
00126
00127  /* Create wind fields (Williamson et al., 1992)... */
00128  for (ix = 0; ix < nx; ix++)
00129      for (iy = 0; iy < ny; iy++)
00130          for (iz = 0; iz < nz; iz++) {
00131              idx = (iz * ny + iy) * nx + ix;
00132              dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00133                                   * (cos(dataLat[iy] * M_PI / 180.0)
00134                                       * cos(alpha * M_PI / 180.0)
00135                                           + sin(dataLat[iy] * M_PI / 180.0)
00136                                               * cos(dataLon[ix] * M_PI / 180.0)
00137                                                   * sin(alpha * M_PI / 180.0)));
00138              dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)

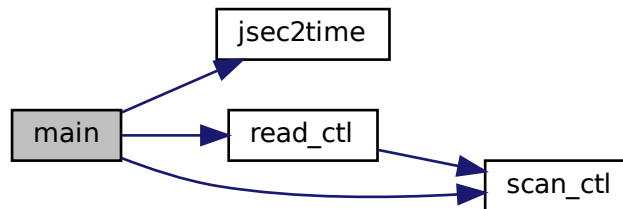
```

```

00139             * sin(dataLon[ix] * M_PI / 180.0)
00140             * sin(alpha * M_PI / 180.0));
00141     dataW[idx] = (float) DZ2DP(1e-3 * w0, dataZ[iz]);
00142 }
00143
00144 /* Write data... */
00145 NC_PUT_FLOAT("T", dataT, 0);
00146 NC_PUT_FLOAT("U", dataU, 0);
00147 NC_PUT_FLOAT("V", dataV, 0);
00148 NC_PUT_FLOAT("W", dataW, 0);
00149
00150 /* Close file... */
00151 NC(nc_close(ncid));
00152
00153 /* Free... */
00154 free(dataT);
00155 free(dataU);
00156 free(dataV);
00157 free(dataW);
00158
00159 return EXIT_SUCCESS;
00160 }

```

Here is the call graph for this function:



5.56 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  Main...
00029  ----- */
00030
00031 int main(
00032     int argc,
00033     char *argv[]) {
00034
00035     ctl_t ctl;
00036
00037     static char filename[LEN];
00038
00039     static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],

```

```

00040     u0, u1, w0, alpha;
00041
00042     static float *dataT, *dataU, *dataV, *dataW;
00043
00044     static int ncid, varid, dims[4], idx, ix, iy, iz, nx, ny, nz,
00045             year, mon, day, hour, min, sec;
00046
00047     static size_t start[4], count[4];
00048
00049     /* Allocate... */
00050     ALLOC(dataT, float,
00051           EP * EY * EX);
00052     ALLOC(dataU, float,
00053           EP * EY * EX);
00054     ALLOC(dataV, float,
00055           EP * EY * EX);
00056     ALLOC(dataW, float,
00057           EP * EY * EX);
00058
00059     /* Check arguments... */
00060     if (argc < 3)
00061         ERRMSG("Give parameters: <ctl> <metbase>");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065     t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00066     nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00067     ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00068     nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00069     z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00070     z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00071     u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00072     u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00073     w0 = scan_ctl(argv[1], argc, argv, "WIND_W0", -1, "0", NULL);
00074     alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00075
00076     /* Check dimensions... */
00077     if (nx < 1 || nx > EX)
00078         ERRMSG("Set 1 <= NX <= MAX!");
00079     if (ny < 1 || ny > EY)
00080         ERRMSG("Set 1 <= NY <= MAX!");
00081     if (nz < 1 || nz > EP)
00082         ERRMSG("Set 1 <= NZ <= MAX!");
00083
00084     /* Get time... */
00085     jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00086     t0 = year * 10000. + mon * 100. + day + hour / 24.;
00087
00088     /* Set filename... */
00089     sprintf(filename, "%s_%d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00090
00091     /* Create netCDF file... */
00092     NC(nc_create(filename, NC_CLOBBER, &ncid));
00093
00094     /* Create dimensions... */
00095     NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00096     NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00097     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00098     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00099
00100     /* Create variables... */
00101     NC_DEF_VAR("time", NC_DOUBLE, 1, &dims[0], "time", "day as %Y%m%d.%f");
00102     NC_DEF_VAR("lev", NC_DOUBLE, 1, &dims[1], "air_pressure", "Pa");
00103     NC_DEF_VAR("lat", NC_DOUBLE, 1, &dims[2], "latitude", "degrees_north");
00104     NC_DEF_VAR("lon", NC_DOUBLE, 1, &dims[3], "longitude", "degrees_east");
00105     NC_DEF_VAR("T", NC_FLOAT, 4, &dims[0], "Temperature", "K");
00106     NC_DEF_VAR("U", NC_FLOAT, 4, &dims[0], "zonal wind", "m s**-1");
00107     NC_DEF_VAR("V", NC_FLOAT, 4, &dims[0], "meridional wind", "m s**-1");
00108     NC_DEF_VAR("W", NC_FLOAT, 4, &dims[0], "vertical velocity", "Pa s**-1");
00109
00110     /* End definition... */
00111     NC(nc_enddef(ncid));
00112
00113     /* Set coordinates... */
00114     for (ix = 0; ix < nx; ix++)
00115         dataLon[ix] = 360.0 / nx * (double) ix;
00116     for (iy = 0; iy < ny; iy++)
00117         dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00118     for (iz = 0; iz < nz; iz++)
00119         dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00120
00121     /* Write coordinates... */
00122     NC_PUT_DOUBLE("time", &t0, 0);
00123     NC_PUT_DOUBLE("lev", dataZ, 0);
00124     NC_PUT_DOUBLE("lat", dataLat, 0);
00125     NC_PUT_DOUBLE("lon", dataLon, 0);
00126

```



```
00127  /* Create wind fields (Williamson et al., 1992)... */
00128  for (ix = 0; ix < nx; ix++)
00129      for (iy = 0; iy < ny; iy++)
00130          for (iz = 0; iz < nz; iz++) {
00131              idx = (iz * ny + iy) * nx + ix;
00132              dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00133                                   * (cos(dataLat[iy] * M_PI / 180.0)
00134                                       * cos(alpha * M_PI / 180.0)
00135                                           + sin(dataLat[iy] * M_PI / 180.0)
00136                                               * cos(dataLon[ix] * M_PI / 180.0)
00137                                                   * sin(alpha * M_PI / 180.0)));
00138              dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00139                                    * sin(dataLon[ix] * M_PI / 180.0)
00140                                        * sin(alpha * M_PI / 180.0));
00141              dataW[idx] = (float) DZ2DP(1e-3 * w0, dataZ[iz]);
00142          }
00143
00144  /* Write data... */
00145  NC_PUT_FLOAT("T", dataT, 0);
00146  NC_PUT_FLOAT("U", dataU, 0);
00147  NC_PUT_FLOAT("V", dataV, 0);
00148  NC_PUT_FLOAT("W", dataW, 0);
00149
00150  /* Close file... */
00151  NC(nc_close(ncid));
00152
00153  /* Free... */
00154  free(dataT);
00155  free(dataU);
00156  free(dataV);
00157  free(dataW);
00158
00159  return EXIT_SUCCESS;
00160 }
```


Index

advect
 ctl_t, 39
ALLOC
 libtrac.h, 300
ARRAY_2D
 libtrac.h, 301
ARRAY_3D
 libtrac.h, 301
atm_basename
 ctl_t, 49
atm_conv.c, 67, 68
 main, 67
atm_dist.c, 69, 74
 main, 70
atm_dt_out
 ctl_t, 49
atm_filter
 ctl_t, 49
atm_gpfile
 ctl_t, 49
atm_init.c, 79, 81
 main, 79
atm_select.c, 83, 86
 main, 83
atm_split.c, 87, 90
 main, 88
atm_stat.c, 92, 95
 main, 92
atm_stride
 ctl_t, 49
atm_t, 3
 lat, 5
 lon, 4
 np, 4
 p, 4
 q, 5
 time, 4
 zeta, 4
atm_type
 ctl_t, 50

balloon
 ctl_t, 39
bound_dps
 ctl_t, 43
bound_dzs
 ctl_t, 43
bound_lat0
 ctl_t, 42
bound_lat1
 ctl_t, 42
bound_mass
 ctl_t, 41
bound_mass_trend
 ctl_t, 41

bound_p0
 ctl_t, 42
bound_p1
 ctl_t, 42
bound_pbl
 ctl_t, 43
bound_vmr
 ctl_t, 42
bound_vmr_trend
 ctl_t, 42
bound_zetas
 ctl_t, 43
buoyancy_frequency
 libtrac.c, 110
 libtrac.h, 317

cache_t, 5
 iso_n, 6
 iso_ps, 6
 iso_ts, 6
 iso_var, 6
 uvwp, 6
cape
 met_t, 64
cape.c, 98, 100
 main, 98
cart2geo
 libtrac.c, 111
 libtrac.h, 317
check_finite
 libtrac.h, 317
chemgrid_lat0
 ctl_t, 46
chemgrid_lat1
 ctl_t, 46
chemgrid_lon0
 ctl_t, 46
chemgrid_lon1
 ctl_t, 46
chemgrid_nx
 ctl_t, 46
chemgrid_ny
 ctl_t, 46
chemgrid_nz
 ctl_t, 45
chemgrid_z0
 ctl_t, 45
chemgrid_z1
 ctl_t, 45
chunkszhint
 ctl_t, 22
cin
 met_t, 64
cl
 met_t, 63

clams_met_data
 ctl_t, 22
 clim_h2o2
 libtrac.c, 119
 libtrac.h, 325
 clim_h2o2_filename
 ctl_t, 44
 clim_h2o2_init
 libtrac.c, 120
 libtrac.h, 326
 clim_hno3
 libtrac.c, 111
 libtrac.h, 317
 clim_hno3_init
 libtrac.c, 112
 libtrac.h, 318
 clim_oh
 libtrac.c, 115
 libtrac.h, 322
 clim_oh_diurnal
 libtrac.c, 116
 libtrac.h, 323
 clim_oh_filename
 ctl_t, 44
 clim_oh_init
 libtrac.c, 117
 libtrac.h, 323
 clim_oh_init_help
 libtrac.c, 118
 libtrac.h, 325
 clim_t, 6
 h2o2, 12
 h2o2_lat, 11
 h2o2_nlat, 11
 h2o2_np, 11
 h2o2_ntime, 11
 h2o2_p, 12
 h2o2_time, 11
 hno3, 9
 hno3_lat, 9
 hno3_nlat, 9
 hno3_np, 9
 hno3_ntime, 8
 hno3_p, 9
 hno3_time, 9
 oh, 11
 oh_lat, 10
 oh_nlat, 10
 oh_np, 10
 oh_ntime, 10
 oh_p, 10
 oh_time, 10
 tropo, 8
 tropo_lat, 8
 tropo_nlat, 8
 tropo_ntime, 8
 tropo_time, 8
 clim_tropo
 libtrac.c, 121
 libtrac.h, 328
 clim_tropo_init
 libtrac.c, 122
 libtrac.h, 328
 compress_pack
 libtrac.c, 124
 libtrac.h, 330
 conv_cape
 ctl_t, 40
 conv_cin
 ctl_t, 40
 conv_dt
 ctl_t, 41
 conv_mix
 ctl_t, 41
 conv_mix_bot
 ctl_t, 41
 conv_mix_top
 ctl_t, 41
 CP
 libtrac.h, 300
 CPD
 libtrac.h, 296
 csi_basename
 ctl_t, 50
 csi_dt_out
 ctl_t, 50
 csi_lat0
 ctl_t, 52
 csi_lat1
 ctl_t, 52
 csi_lon0
 ctl_t, 51
 csi_lon1
 ctl_t, 51
 csi_modmin
 ctl_t, 50
 csi_nx
 ctl_t, 51
 csi_ny
 ctl_t, 52
 csi_nz
 ctl_t, 51
 csi_obsfile
 ctl_t, 50
 csi_obsmin
 ctl_t, 50
 csi_z0
 ctl_t, 51
 csi_z1
 ctl_t, 51
 CT
 libtrac.h, 300
 ctl_t, 12
 advect, 39
 atm_basename, 49
 atm_dt_out, 49

atm_filter, [49](#)
atm_gpfile, [49](#)
atm_stride, [49](#)
atm_type, [50](#)
balloon, [39](#)
bound_dps, [43](#)
bound_dzs, [43](#)
bound_lat0, [42](#)
bound_lat1, [42](#)
bound_mass, [41](#)
bound_mass_trend, [41](#)
bound_p0, [42](#)
bound_p1, [42](#)
bound_pbl, [43](#)
bound_vmr, [42](#)
bound_vmr_trend, [42](#)
bound_zetas, [43](#)
chemgrid_lat0, [46](#)
chemgrid_lat1, [46](#)
chemgrid_lon0, [46](#)
chemgrid_lon1, [46](#)
chemgrid_nx, [46](#)
chemgrid_ny, [46](#)
chemgrid_nz, [45](#)
chemgrid_z0, [45](#)
chemgrid_z1, [45](#)
chunkszhint, [22](#)
clams_met_data, [22](#)
clim_h2o2_filename, [44](#)
clim_oh_filename, [44](#)
conv_cape, [40](#)
conv_cin, [40](#)
conv_dt, [41](#)
conv_mix, [41](#)
conv_mix_bot, [41](#)
conv_mix_top, [41](#)
csi_basename, [50](#)
csi_dt_out, [50](#)
csi_lat0, [52](#)
csi_lat1, [52](#)
csi_lon0, [51](#)
csi_lon1, [51](#)
csi_modmin, [50](#)
csi_nx, [51](#)
csi_ny, [52](#)
csi_nz, [51](#)
csi_obsfile, [50](#)
csi_obsmin, [50](#)
csi_z0, [51](#)
csi_z1, [51](#)
direction, [33](#)
dry_depo_dp, [47](#)
dry_depo_vdep, [47](#)
dt_met, [34](#)
dt_mod, [34](#)
ens_basename, [52](#)
ens_dt_out, [52](#)
grid_basename, [52](#)
grid_dt_out, [53](#)
grid_gpfile, [53](#)
grid_lat0, [54](#)
grid_lat1, [54](#)
grid_lon0, [54](#)
grid_lon1, [54](#)
grid_nx, [54](#)
grid_ny, [54](#)
grid_nz, [53](#)
grid_sparse, [53](#)
grid_type, [55](#)
grid_z0, [53](#)
grid_z1, [53](#)
h2o2_chem_cc, [45](#)
h2o2_chem_reaction, [45](#)
isosurf, [39](#)
met_cache, [38](#)
met_cloud, [38](#)
met_cloud_min, [38](#)
met_detrend, [35](#)
met_dp, [35](#)
met_dt_out, [38](#)
met_dx, [34](#)
met_dy, [35](#)
met_geopot_sx, [36](#)
met_geopot_sy, [36](#)
met_nc_scale, [34](#)
met_np, [36](#)
met_p, [36](#)
met_relhum, [36](#)
met_sp, [35](#)
met_sx, [35](#)
met_sy, [35](#)
met_tropo, [36](#)
met_tropo_lapse, [37](#)
met_tropo_lapse_sep, [37](#)
met_tropo_nlev, [37](#)
met_tropo_nlev_sep, [37](#)
met_tropo_pv, [37](#)
met_tropo_spline, [38](#)
met_tropo_theta, [37](#)
met_type, [34](#)
metbase, [34](#)
molmass, [43](#)
nq, [22](#)
oh_chem, [44](#)
oh_chem_beta, [45](#)
oh_chem_reaction, [44](#)
prof_basename, [55](#)
prof_lat0, [56](#)
prof_lat1, [56](#)
prof_lon0, [56](#)
prof_lon1, [56](#)
prof_nx, [56](#)
prof_ny, [56](#)
prof_nz, [55](#)
prof_obsfile, [55](#)
prof_z0, [55](#)

prof_z1, 55
 psc_h2o, 48
 psc_hno3, 49
 qnt_cape, 29
 qnt_cin, 29
 qnt_cl, 28
 qnt_ens, 23
 qnt_format, 23
 qnt_h2o, 27
 qnt_h2ot, 26
 qnt_hno3, 29
 qnt_idx, 23
 qnt_iwc, 27
 qnt_lapse, 32
 qnt_longname, 22
 qnt_lwc, 27
 qnt_m, 23
 qnt_mloss_decay, 30
 qnt_mloss_dry, 30
 qnt_mloss_h2o2, 30
 qnt_mloss_oh, 29
 qnt_mloss_wet, 30
 qnt_name, 22
 qnt_o3, 27
 qnt_oh, 29
 qnt_p, 26
 qnt_pbl, 25
 qnt_pcb, 28
 qnt_pct, 28
 qnt_pel, 28
 qnt_plcl, 28
 qnt_plfc, 28
 qnt_ps, 24
 qnt_psat, 30
 qnt_psice, 30
 qnt_pt, 25
 qnt_pv, 32
 qnt_pw, 31
 qnt_rh, 31
 qnt_rhice, 31
 qnt_rho, 26
 qnt_rhop, 24
 qnt_rp, 24
 qnt_sh, 31
 qnt_stat, 23
 qnt_t, 26
 qnt_tdew, 32
 qnt_theta, 31
 qnt_tice, 33
 qnt_tnat, 33
 qnt_ts, 24
 qnt_tsts, 33
 qnt_tt, 25
 qnt_tvirt, 32
 qnt_u, 26
 qnt_unit, 23
 qnt_us, 25
 qnt_v, 27
 qnt_vh, 32
 qnt_vmr, 24
 qnt_vmrimpl, 29
 qnt_vs, 25
 qnt_vz, 32
 qnt_w, 27
 qnt_z, 26
 qnt_zeta, 31
 qnt_zs, 24
 qnt_zt, 25
 read_mode, 22
 reflect, 39
 sample_basename, 57
 sample_dx, 57
 sample_dz, 57
 sample_obsfile, 57
 sort_dt, 38
 species, 43
 stat_basename, 57
 stat_lat, 58
 stat_lon, 57
 stat_r, 58
 stat_t0, 58
 stat_t1, 58
 t_start, 33
 t_stop, 33
 tdec_strat, 44
 tdec_trop, 44
 turb_dx_strat, 39
 turb_dx_trop, 39
 turb_dz_strat, 40
 turb_dz_trop, 40
 turb_mesox, 40
 turb_mesoz, 40
 vert_coord_ap, 21
 vert_coord_met, 21
 vert_vel, 21
 wet_depo_bc_a, 47
 wet_depo_bc_b, 47
 wet_depo_bc_h, 48
 wet_depo_bc_ret_ratio, 48
 wet_depo_ic_a, 47
 wet_depo_ic_b, 48
 wet_depo_ic_h, 48
 wet_depo_ic_ret_ratio, 48
 wet_depo_pre, 47
 CY
 libtrac.h, 300
 day2doy
 libtrac.c, 125
 libtrac.h, 331
 day2doy.c, 102, 103
 main, 102
 DEG2DX
 libtrac.h, 301
 DEG2DY
 libtrac.h, 301
 direction

- ctl_t, 33
- DIST
 - libtrac.h, 302
- DIST2
 - libtrac.h, 302
- DLAPSE
 - met_lapse.c, 440
- DOTP
 - libtrac.h, 303
- doy2day
 - libtrac.c, 126
 - libtrac.h, 332
- doy2day.c, 104, 105
 - main, 104
- DP2DZ
 - libtrac.h, 301
- dry_depo_dp
 - ctl_t, 47
- dry_depo_vdep
 - ctl_t, 47
- dt_met
 - ctl_t, 34
- dt_mod
 - ctl_t, 34
- DX2DEG
 - libtrac.h, 302
- DY2DEG
 - libtrac.h, 302
- DZ2DP
 - libtrac.h, 302
- ens_basename
 - ctl_t, 52
- ens_dt_out
 - ctl_t, 52
- EP
 - libtrac.h, 299
- EPS
 - libtrac.h, 296
- ERRMSG
 - libtrac.h, 315
- EX
 - libtrac.h, 299
- EY
 - libtrac.h, 299
- fft_help
 - met_spec.c, 470
- FMOD
 - libtrac.h, 303
- FREAD
 - libtrac.h, 303
- FWRITE
 - libtrac.h, 303
- G0
 - libtrac.h, 296
- geo2cart
 - libtrac.c, 126
- libtrac.h, 332
- get_met
 - libtrac.c, 126
 - libtrac.h, 333
- get_met_help
 - libtrac.c, 129
 - libtrac.h, 335
- get_met_replace
 - libtrac.c, 130
 - libtrac.h, 336
- get_tropo
 - tropo.c, 552
- grid_basename
 - ctl_t, 52
- grid_dt_out
 - ctl_t, 53
- grid_gpfiler
 - ctl_t, 53
- grid_lat0
 - ctl_t, 54
- grid_lat1
 - ctl_t, 54
- grid_lon0
 - ctl_t, 54
- grid_lon1
 - ctl_t, 54
- grid_nx
 - ctl_t, 54
- grid_ny
 - ctl_t, 54
- grid_nz
 - ctl_t, 53
- grid_sparse
 - ctl_t, 53
- grid_type
 - ctl_t, 55
- grid_z0
 - ctl_t, 53
- grid_z1
 - ctl_t, 53
- H0
 - libtrac.h, 296
- h2o
 - met_t, 65
- h2o2
 - clim_t, 12
- h2o2_chem_cc
 - ctl_t, 45
- h2o2_chem_reaction
 - ctl_t, 45
- h2o2_lat
 - clim_t, 11
- h2o2_nlat
 - clim_t, 11
- h2o2_np
 - clim_t, 11
- h2o2_ntime
 - clim_t, 11

- h2o2_p
 - clim_t, [12](#)
- h2o2_time
 - clim_t, [11](#)
- h2ot
 - met_t, [63](#)
- hno3
 - clim_t, [9](#)
- hno3_lat
 - clim_t, [9](#)
- hno3_nlat
 - clim_t, [9](#)
- hno3_np
 - clim_t, [9](#)
- hno3_ntime
 - clim_t, [8](#)
- hno3_p
 - clim_t, [9](#)
- hno3_time
 - clim_t, [9](#)
- IDXMAX
 - met_lapse.c, [440](#)
- INTPOL_2D
 - libtrac.h, [304](#)
- INTPOL_3D
 - libtrac.h, [304](#)
- INTPOL_INIT
 - libtrac.h, [304](#)
- intpol_met_space_2d
 - libtrac.c, [132](#)
 - libtrac.h, [338](#)
- intpol_met_space_3d
 - libtrac.c, [131](#)
 - libtrac.h, [337](#)
- intpol_met_time_2d
 - libtrac.c, [134](#)
 - libtrac.h, [340](#)
- intpol_met_time_3d
 - libtrac.c, [133](#)
 - libtrac.h, [339](#)
- INTPOL_SPACE_ALL
 - libtrac.h, [304](#)
- INTPOL_TIME_ALL
 - libtrac.h, [305](#)
- intpol_tropo_3d
 - tropo_sample.c, [562](#)
- iso_n
 - cache_t, [6](#)
- iso_ps
 - cache_t, [6](#)
- iso_ts
 - cache_t, [6](#)
- iso_var
 - cache_t, [6](#)
- isosurf
 - ctl_t, [39](#)
- iwc
 - met_t, [66](#)
- jsec2time
 - libtrac.c, [134](#)
 - libtrac.h, [340](#)
- jsec2time.c, [105](#), [106](#)
 - main, [105](#)
- KB
 - libtrac.h, [297](#)
- LAPSE
 - libtrac.h, [306](#)
- lapse_rate
 - libtrac.c, [135](#)
 - libtrac.h, [341](#)
- LAPSEMIN
 - met_lapse.c, [440](#)
- lat
 - atm_t, [5](#)
 - met_t, [61](#)
- LEN
 - libtrac.h, [298](#)
- libtrac.c, [107](#), [209](#)
 - buoyancy_frequency, [110](#)
 - cart2geo, [111](#)
 - clim_h2o2, [119](#)
 - clim_h2o2_init, [120](#)
 - clim_hno3, [111](#)
 - clim_hno3_init, [112](#)
 - clim_oh, [115](#)
 - clim_oh_diurnal, [116](#)
 - clim_oh_init, [117](#)
 - clim_oh_init_help, [118](#)
 - clim_tropo, [121](#)
 - clim_tropo_init, [122](#)
 - compress_pack, [124](#)
 - day2doy, [125](#)
 - doy2day, [126](#)
 - geo2cart, [126](#)
 - get_met, [126](#)
 - get_met_help, [129](#)
 - get_met_replace, [130](#)
 - intpol_met_space_2d, [132](#)
 - intpol_met_space_3d, [131](#)
 - intpol_met_time_2d, [134](#)
 - intpol_met_time_3d, [133](#)
 - jsec2time, [134](#)
 - lapse_rate, [135](#)
 - locate_irr, [135](#)
 - locate_reg, [136](#)
 - nat_temperature, [136](#)
 - quicksort, [137](#)
 - quicksort_partition, [137](#)
 - read_atm, [138](#)
 - read_atm_asc, [139](#)
 - read_atm_bin, [140](#)
 - read_atm_clams, [140](#)
 - read_atm_nc, [141](#)
 - read_clim, [142](#)
 - read_ctl, [142](#)

- read_met, [150](#)
- read_met_bin_2d, [153](#)
- read_met_bin_3d, [154](#)
- read_met_cape, [155](#)
- read_met_cloud, [156](#)
- read_met_detrend, [157](#)
- read_met_extrapolate, [159](#)
- read_met_geopot, [159](#)
- read_met_grid, [161](#)
- read_met_levels, [163](#)
- read_met_ml2pl, [165](#)
- read_met_nc_2d, [166](#)
- read_met_nc_3d, [168](#)
- read_met_pbl, [169](#)
- read_met_periodic, [170](#)
- read_met_pv, [171](#)
- read_met_sample, [172](#)
- read_met_surface, [174](#)
- read_met_tropo, [175](#)
- read_obs, [178](#)
- scan_ctl, [178](#)
- sedi, [179](#)
- spline, [180](#)
- stddev, [181](#)
- sza, [181](#)
- time2jsec, [182](#)
- timer, [182](#)
- tropo_weight, [183](#)
- write_atm, [184](#)
- write_atm_asc, [185](#)
- write_atm_bin, [186](#)
- write_atm_clams, [187](#)
- write_atm_nc, [189](#)
- write_csi, [190](#)
- write_ens, [193](#)
- write_grid, [194](#)
- write_grid_asc, [197](#)
- write_grid_nc, [198](#)
- write_met, [200](#)
- write_met_bin_2d, [201](#)
- write_met_bin_3d, [202](#)
- write_prof, [203](#)
- write_sample, [206](#)
- write_station, [208](#)
- libtrac.h, [288](#), [415](#)
 - ALLOC, [300](#)
 - ARRAY_2D, [301](#)
 - ARRAY_3D, [301](#)
 - buoyancy_frequency, [317](#)
 - cart2geo, [317](#)
 - check_finite, [317](#)
 - clim_h2o2, [325](#)
 - clim_h2o2_init, [326](#)
 - clim_hno3, [317](#)
 - clim_hno3_init, [318](#)
 - clim_oh, [322](#)
 - clim_oh_diurnal, [323](#)
 - clim_oh_init, [323](#)
 - clim_oh_init_help, [325](#)
 - clim_tropo, [328](#)
 - clim_tropo_init, [328](#)
 - compress_pack, [330](#)
 - CP, [300](#)
 - CPD, [296](#)
 - CT, [300](#)
 - CY, [300](#)
 - day2doy, [331](#)
 - DEG2DX, [301](#)
 - DEG2DY, [301](#)
 - DIST, [302](#)
 - DIST2, [302](#)
 - DOTP, [303](#)
 - doy2day, [332](#)
 - DP2DZ, [301](#)
 - DX2DEG, [302](#)
 - DY2DEG, [302](#)
 - DZ2DP, [302](#)
 - EP, [299](#)
 - EPS, [296](#)
 - ERRMSG, [315](#)
 - EX, [299](#)
 - EY, [299](#)
 - FMOD, [303](#)
 - FREAD, [303](#)
 - FWRITE, [303](#)
 - G0, [296](#)
 - geo2cart, [332](#)
 - get_met, [333](#)
 - get_met_help, [335](#)
 - get_met_replace, [336](#)
 - H0, [296](#)
 - INTPOL_2D, [304](#)
 - INTPOL_3D, [304](#)
 - INTPOL_INIT, [304](#)
 - intpol_met_space_2d, [338](#)
 - intpol_met_space_3d, [337](#)
 - intpol_met_time_2d, [340](#)
 - intpol_met_time_3d, [339](#)
 - INTPOL_SPACE_ALL, [304](#)
 - INTPOL_TIME_ALL, [305](#)
 - jsec2time, [340](#)
 - KB, [297](#)
 - LAPSE, [306](#)
 - lapse_rate, [341](#)
 - LEN, [298](#)
 - LIN, [306](#)
 - locate_irr, [341](#)
 - locate_reg, [342](#)
 - LOG, [314](#)
 - LOGLEV, [314](#)
 - LV, [297](#)
 - MA, [297](#)
 - MH2O, [297](#)
 - MO3, [297](#)
 - nat_temperature, [342](#)
 - NC, [306](#)

NC_DEF_VAR, 306
 NC_GET_DOUBLE, 307
 NC_INQ_DIM, 307
 NC_PUT_ATT, 308
 NC_PUT_ATT_GLOBAL, 308
 NC_PUT_DOUBLE, 307
 NC_PUT_FLOAT, 309
 NC_PUT_INT, 308
 NCSI, 299
 NENS, 299
 NN, 309
 NOBS, 300
 NORM, 309
 NP, 298
 NQ, 299
 NTHREADS, 300
 NTIMER, 315
 NVTX_POP, 316
 NVTX_PUSH, 316
 P, 309
 P0, 297
 PRINT, 315
 PRINT_TIMERS, 315
 PSAT, 310
 PSICE, 310
 PW, 310
 quicksort, 343
 quicksort_partition, 343
 RA, 298
 RE, 298
 read_atm, 344
 read_atm_asc, 345
 read_atm_bin, 346
 read_atm_clams, 346
 read_atm_nc, 347
 read_clim, 348
 read_ctl, 348
 read_met, 356
 read_met_bin_2d, 359
 read_met_bin_3d, 360
 read_met_cape, 361
 read_met_cloud, 362
 read_met_detrend, 363
 read_met_extrapolate, 365
 read_met_geopot, 365
 read_met_grid, 367
 read_met_levels, 369
 read_met_ml2pl, 371
 read_met_nc_2d, 372
 read_met_nc_3d, 374
 read_met_pbl, 375
 read_met_periodic, 376
 read_met_pv, 377
 read_met_sample, 378
 read_met_surface, 380
 read_met_tropo, 381
 read_obs, 384
 RH, 310
 RHICE, 310
 RHO, 311
 RI, 298
 scan_ctl, 384
 sedi, 385
 SELECT_TIMER, 315
 SET_ATM, 311
 SET_QNT, 311
 SH, 311
 spline, 386
 SQR, 312
 START_TIMERS, 316
 stddev, 387
 STOP_TIMERS, 316
 SWAP, 312
 sza, 387
 T0, 298
 TDEW, 312
 THETA, 312
 THETA_VIRT, 313
 thrustSortWrapper, 316
 TICE, 312
 time2jsec, 388
 timer, 388
 TOK, 313
 tropo_weight, 389
 TVIRT, 313
 WARN, 314
 write_atm, 390
 write_atm_asc, 391
 write_atm_bin, 392
 write_atm_clams, 393
 write_atm_nc, 395
 write_csi, 396
 write_ens, 399
 write_grid, 400
 write_grid_asc, 403
 write_grid_nc, 404
 write_met, 406
 write_met_bin_2d, 407
 write_met_bin_3d, 408
 write_prof, 409
 write_sample, 412
 write_station, 414
 Z, 313
 ZDIFF, 313
 ZETA, 314
 LIN
 libtrac.h, 306
 locate_irr
 libtrac.c, 135
 libtrac.h, 341
 locate_reg
 libtrac.c, 136
 libtrac.h, 342
 LOG
 libtrac.h, 314
 LOGLEV

- libtrac.h, 314
- lon
 - atm_t, 4
 - met_t, 61
- LV
 - libtrac.h, 297
- lwc
 - met_t, 66
- MA
 - libtrac.h, 297
- main
 - atm_conv.c, 67
 - atm_dist.c, 70
 - atm_init.c, 79
 - atm_select.c, 83
 - atm_split.c, 88
 - atm_stat.c, 92
 - cape.c, 98
 - day2doy.c, 102
 - doy2day.c, 104
 - jsec2time.c, 105
 - met_conv.c, 437
 - met_lapse.c, 440
 - met_map.c, 448
 - met_prof.c, 456
 - met_sample.c, 464
 - met_spec.c, 471
 - met_subgrid.c, 476
 - met_zm.c, 483
 - sedi.c, 490
 - time2jsec.c, 492
 - tnat.c, 494
 - trac.c, 522
 - tropo.c, 553
 - tropo_sample.c, 563
 - tropo_zm.c, 571
 - wind.c, 577
- met_cache
 - ctl_t, 38
- met_cloud
 - ctl_t, 38
- met_cloud_min
 - ctl_t, 38
- met_conv.c, 436, 438
 - main, 437
- met_detrend
 - ctl_t, 35
- met_dp
 - ctl_t, 35
- met_dt_out
 - ctl_t, 38
- met_dx
 - ctl_t, 34
- met_dy
 - ctl_t, 35
- met_geopot_sx
 - ctl_t, 36
- met_geopot_sy
 - ctl_t, 36
- met_lapse.c, 439, 444
 - DLAPSE, 440
 - IDXMAX, 440
 - LAPSEMIN, 440
 - main, 440
- met_map.c, 447, 452
 - main, 448
 - NX, 448
 - NY, 448
- met_nc_scale
 - ctl_t, 34
- met_np
 - ctl_t, 36
- met_p
 - ctl_t, 36
- met_prof.c, 456, 460
 - main, 456
 - NZ, 456
- met_relhum
 - ctl_t, 36
- met_sample.c, 463, 467
 - main, 464
- met_sp
 - ctl_t, 35
- met_spec.c, 470, 473
 - fft_help, 470
 - main, 471
 - PMAX, 470
- met_subgrid.c, 475, 479
 - main, 476
- met_sx
 - ctl_t, 35
- met_sy
 - ctl_t, 35
- met_t, 58
 - cape, 64
 - cin, 64
 - cl, 63
 - h2o, 65
 - h2ot, 63
 - iwc, 66
 - lat, 61
 - lon, 61
 - lwc, 66
 - np, 61
 - nx, 60
 - ny, 61
 - o3, 66
 - p, 61
 - patp, 66
 - pbl, 62
 - pcb, 63
 - pct, 63
 - pel, 64
 - pl, 66
 - plcl, 64
 - plfc, 64

- ps, [61](#)
- pt, [62](#)
- pv, [65](#)
- t, [65](#)
- time, [60](#)
- ts, [62](#)
- tt, [63](#)
- u, [65](#)
- us, [62](#)
- v, [65](#)
- vs, [62](#)
- w, [65](#)
- z, [64](#)
- zeta, [66](#)
- zeta_dot, [67](#)
- zs, [62](#)
- zt, [63](#)
- met_tropo
 - ctl_t, [36](#)
- met_tropo_lapse
 - ctl_t, [37](#)
- met_tropo_lapse_sep
 - ctl_t, [37](#)
- met_tropo_nlev
 - ctl_t, [37](#)
- met_tropo_nlev_sep
 - ctl_t, [37](#)
- met_tropo_pv
 - ctl_t, [37](#)
- met_tropo_spline
 - ctl_t, [38](#)
- met_tropo_theta
 - ctl_t, [37](#)
- met_type
 - ctl_t, [34](#)
- met_zm.c, [482](#), [486](#)
 - main, [483](#)
 - NY, [482](#)
 - NZ, [482](#)
- metbase
 - ctl_t, [34](#)
- MH2O
 - libtrac.h, [297](#)
- MO3
 - libtrac.h, [297](#)
- module_advect
 - trac.c, [496](#)
- module_bound_cond
 - trac.c, [498](#)
- module_chemgrid
 - trac.c, [511](#)
- module_convection
 - trac.c, [499](#)
- module_decay
 - trac.c, [500](#)
- module_diffusion_meso
 - trac.c, [501](#)
- module_diffusion_turb
 - trac.c, [503](#)
- module_dry_deposition
 - trac.c, [504](#)
- module_h2o2_chem
 - trac.c, [510](#)
- module_isosurf
 - trac.c, [506](#)
- module_isosurf_init
 - trac.c, [505](#)
- module_meteo
 - trac.c, [507](#)
- module_oh_chem
 - trac.c, [508](#)
- module_position
 - trac.c, [513](#)
- module_rng
 - trac.c, [514](#)
- module_rng_init
 - trac.c, [514](#)
- module_sedi
 - trac.c, [515](#)
- module_sort
 - trac.c, [516](#)
- module_sort_help
 - trac.c, [517](#)
- module_timesteps
 - trac.c, [518](#)
- module_timesteps_init
 - trac.c, [518](#)
- module_wet_deposition
 - trac.c, [519](#)
- molmass
 - ctl_t, [43](#)
- nat_temperature
 - libtrac.c, [136](#)
 - libtrac.h, [342](#)
- NC
 - libtrac.h, [306](#)
- NC_DEF_VAR
 - libtrac.h, [306](#)
- NC_GET_DOUBLE
 - libtrac.h, [307](#)
- NC_INQ_DIM
 - libtrac.h, [307](#)
- NC_PUT_ATT
 - libtrac.h, [308](#)
- NC_PUT_ATT_GLOBAL
 - libtrac.h, [308](#)
- NC_PUT_DOUBLE
 - libtrac.h, [307](#)
- NC_PUT_FLOAT
 - libtrac.h, [309](#)
- NC_PUT_INT
 - libtrac.h, [308](#)
- NCSI
 - libtrac.h, [299](#)
- NENS
 - libtrac.h, [299](#)

NN
 libtrac.h, 309

NOBS
 libtrac.h, 300

NORM
 libtrac.h, 309

NP
 libtrac.h, 298

np
 atm_t, 4
 met_t, 61

NQ
 libtrac.h, 299

nq
 ctl_t, 22

NT
 tropo_sample.c, 562
 tropo_zm.c, 571

NTHREADS
 libtrac.h, 300

NTIMER
 libtrac.h, 315

NVTX_POP
 libtrac.h, 316

NVTX_PUSH
 libtrac.h, 316

NX
 met_map.c, 448

nx
 met_t, 60

NY
 met_map.c, 448
 met_zm.c, 482

ny
 met_t, 61

NZ
 met_prof.c, 456
 met_zm.c, 482

o3
 met_t, 66

oh
 clim_t, 11

oh_chem
 ctl_t, 44

oh_chem_beta
 ctl_t, 45

oh_chem_reaction
 ctl_t, 44

oh_lat
 clim_t, 10

oh_nlat
 clim_t, 10

oh_np
 clim_t, 10

oh_nptime
 clim_t, 10

oh_p
 clim_t, 10

oh_time
 clim_t, 10

P
 libtrac.h, 309

p
 atm_t, 4
 met_t, 61

P0
 libtrac.h, 297

patp
 met_t, 66

pbl
 met_t, 62

pcb
 met_t, 63

pct
 met_t, 63

pel
 met_t, 64

pl
 met_t, 66

plcl
 met_t, 64

plfc
 met_t, 64

PMAX
 met_spec.c, 470

PRINT
 libtrac.h, 315

PRINT_TIMERS
 libtrac.h, 315

prof_basename
 ctl_t, 55

prof_lat0
 ctl_t, 56

prof_lat1
 ctl_t, 56

prof_lon0
 ctl_t, 56

prof_lon1
 ctl_t, 56

prof_nx
 ctl_t, 56

prof_ny
 ctl_t, 56

prof_nz
 ctl_t, 55

prof_obsfile
 ctl_t, 55

prof_z0
 ctl_t, 55

prof_z1
 ctl_t, 55

ps
 met_t, 61

PSAT
 libtrac.h, 310

psc_h2o

ctl_t, 48
 psc_hno3
 ctl_t, 49
 PSICE
 libtrac.h, 310
 pt
 met_t, 62
 pv
 met_t, 65
 PW
 libtrac.h, 310
 q
 atm_t, 5
 qnt_cape
 ctl_t, 29
 qnt_cin
 ctl_t, 29
 qnt_cl
 ctl_t, 28
 qnt_ens
 ctl_t, 23
 qnt_format
 ctl_t, 23
 qnt_h2o
 ctl_t, 27
 qnt_h2ot
 ctl_t, 26
 qnt_hno3
 ctl_t, 29
 qnt_idx
 ctl_t, 23
 qnt_iwc
 ctl_t, 27
 qnt_lapse
 ctl_t, 32
 qnt_longname
 ctl_t, 22
 qnt_lwc
 ctl_t, 27
 qnt_m
 ctl_t, 23
 qnt_mloss_decay
 ctl_t, 30
 qnt_mloss_dry
 ctl_t, 30
 qnt_mloss_h2o2
 ctl_t, 30
 qnt_mloss_oh
 ctl_t, 29
 qnt_mloss_wet
 ctl_t, 30
 qnt_name
 ctl_t, 22
 qnt_o3
 ctl_t, 27
 qnt_oh
 ctl_t, 29
 qnt_p
 ctl_t, 26
 qnt_pbl
 ctl_t, 25
 qnt_pcb
 ctl_t, 28
 qnt_pct
 ctl_t, 28
 qnt_pel
 ctl_t, 28
 qnt_plcl
 ctl_t, 28
 qnt_plfc
 ctl_t, 28
 qnt_ps
 ctl_t, 24
 qnt_psat
 ctl_t, 30
 qnt_psice
 ctl_t, 30
 qnt_pt
 ctl_t, 25
 qnt_pv
 ctl_t, 32
 qnt_pw
 ctl_t, 31
 qnt_rh
 ctl_t, 31
 qnt_rhice
 ctl_t, 31
 qnt_rho
 ctl_t, 26
 qnt_rhop
 ctl_t, 24
 qnt_rp
 ctl_t, 24
 qnt_sh
 ctl_t, 31
 qnt_stat
 ctl_t, 23
 qnt_t
 ctl_t, 26
 qnt_tdew
 ctl_t, 32
 qnt_theta
 ctl_t, 31
 qnt_tice
 ctl_t, 33
 qnt_tnat
 ctl_t, 33
 qnt_ts
 ctl_t, 24
 qnt_tsts
 ctl_t, 33
 qnt_tt
 ctl_t, 25
 qnt_tvirt
 ctl_t, 32
 qnt_u

- ctl_t, 26
- qnt_unit
 - ctl_t, 23
- qnt_us
 - ctl_t, 25
- qnt_v
 - ctl_t, 27
- qnt_vh
 - ctl_t, 32
- qnt_vmr
 - ctl_t, 24
- qnt_vmrimpl
 - ctl_t, 29
- qnt_vs
 - ctl_t, 25
- qnt_vz
 - ctl_t, 32
- qnt_w
 - ctl_t, 27
- qnt_z
 - ctl_t, 26
- qnt_zeta
 - ctl_t, 31
- qnt_zs
 - ctl_t, 24
- qnt_zt
 - ctl_t, 25
- quicksort
 - libtrac.c, 137
 - libtrac.h, 343
- quicksort_partition
 - libtrac.c, 137
 - libtrac.h, 343
- RA
 - libtrac.h, 298
- RE
 - libtrac.h, 298
- read_atm
 - libtrac.c, 138
 - libtrac.h, 344
- read_atm_asc
 - libtrac.c, 139
 - libtrac.h, 345
- read_atm_bin
 - libtrac.c, 140
 - libtrac.h, 346
- read_atm_clams
 - libtrac.c, 140
 - libtrac.h, 346
- read_atm_nc
 - libtrac.c, 141
 - libtrac.h, 347
- read_clim
 - libtrac.c, 142
 - libtrac.h, 348
- read_ctl
 - libtrac.c, 142
 - libtrac.h, 348
- read_met
 - libtrac.c, 150
 - libtrac.h, 356
- read_met_bin_2d
 - libtrac.c, 153
 - libtrac.h, 359
- read_met_bin_3d
 - libtrac.c, 154
 - libtrac.h, 360
- read_met_cape
 - libtrac.c, 155
 - libtrac.h, 361
- read_met_cloud
 - libtrac.c, 156
 - libtrac.h, 362
- read_met_detrend
 - libtrac.c, 157
 - libtrac.h, 363
- read_met_extrapolate
 - libtrac.c, 159
 - libtrac.h, 365
- read_met_geopot
 - libtrac.c, 159
 - libtrac.h, 365
- read_met_grid
 - libtrac.c, 161
 - libtrac.h, 367
- read_met_levels
 - libtrac.c, 163
 - libtrac.h, 369
- read_met_ml2pl
 - libtrac.c, 165
 - libtrac.h, 371
- read_met_nc_2d
 - libtrac.c, 166
 - libtrac.h, 372
- read_met_nc_3d
 - libtrac.c, 168
 - libtrac.h, 374
- read_met_pbl
 - libtrac.c, 169
 - libtrac.h, 375
- read_met_periodic
 - libtrac.c, 170
 - libtrac.h, 376
- read_met_pv
 - libtrac.c, 171
 - libtrac.h, 377
- read_met_sample
 - libtrac.c, 172
 - libtrac.h, 378
- read_met_surface
 - libtrac.c, 174
 - libtrac.h, 380
- read_met_tropo
 - libtrac.c, 175
 - libtrac.h, 381
- read_mode

- ctl_t, 22
- read_obs
 - libtrac.c, 178
 - libtrac.h, 384
- reflect
 - ctl_t, 39
- RH
 - libtrac.h, 310
- RHICE
 - libtrac.h, 310
- RHO
 - libtrac.h, 311
- RI
 - libtrac.h, 298
- sample_basename
 - ctl_t, 57
- sample_dx
 - ctl_t, 57
- sample_dz
 - ctl_t, 57
- sample_obsfile
 - ctl_t, 57
- scan_ctl
 - libtrac.c, 178
 - libtrac.h, 384
- sedi
 - libtrac.c, 179
 - libtrac.h, 385
- sedi.c, 490, 491
 - main, 490
- SELECT_TIMER
 - libtrac.h, 315
- SET_ATM
 - libtrac.h, 311
- SET_QNT
 - libtrac.h, 311
- SH
 - libtrac.h, 311
- sort_dt
 - ctl_t, 38
- species
 - ctl_t, 43
- spline
 - libtrac.c, 180
 - libtrac.h, 386
- SQR
 - libtrac.h, 312
- START_TIMERS
 - libtrac.h, 316
- stat_basename
 - ctl_t, 57
- stat_lat
 - ctl_t, 58
- stat_lon
 - ctl_t, 57
- stat_r
 - ctl_t, 58
- stat_t0
 - ctl_t, 58
- stat_t1
 - ctl_t, 58
- stddev
 - libtrac.c, 181
 - libtrac.h, 387
- STOP_TIMERS
 - libtrac.h, 316
- SWAP
 - libtrac.h, 312
- sza
 - libtrac.c, 181
 - libtrac.h, 387
- t
 - met_t, 65
- T0
 - libtrac.h, 298
- t_start
 - ctl_t, 33
- t_stop
 - ctl_t, 33
- tdec_strat
 - ctl_t, 44
- tdec_trop
 - ctl_t, 44
- TDEW
 - libtrac.h, 312
- THETA
 - libtrac.h, 312
- THETAVIRT
 - libtrac.h, 313
- thrustSortWrapper
 - libtrac.h, 316
- TICE
 - libtrac.h, 312
- time
 - atm_t, 4
 - met_t, 60
- time2jsec
 - libtrac.c, 182
 - libtrac.h, 388
- time2jsec.c, 492, 493
 - main, 492
- timer
 - libtrac.c, 182
 - libtrac.h, 388
- tnat.c, 493, 495
 - main, 494
- TOK
 - libtrac.h, 313
- trac.c, 495, 527
 - main, 522
 - module_advect, 496
 - module_bound_cond, 498
 - module_chemgrid, 511
 - module_convection, 499
 - module_decay, 500
 - module_diffusion_meso, 501

- module_diffusion_turb, 503
- module_dry_deposition, 504
- module_h2o2_chem, 510
- module_isosurf, 506
- module_isosurf_init, 505
- module_meteo, 507
- module_oh_chem, 508
- module_position, 513
- module_rng, 514
- module_rng_init, 514
- module_sedi, 515
- module_sort, 516
- module_sort_help, 517
- module_timesteps, 518
- module_timesteps_init, 518
- module_wet_deposition, 519
- write_output, 521
- tropo
 - clim_t, 8
- tropo.c, 552, 557
 - get_tropo, 552
 - main, 553
- tropo_lat
 - clim_t, 8
- tropo_nlat
 - clim_t, 8
- tropo_ntime
 - clim_t, 8
- tropo_sample.c, 561, 566
 - intpol_tropo_3d, 562
 - main, 563
 - NT, 562
- tropo_time
 - clim_t, 8
- tropo_weight
 - libtrac.c, 183
 - libtrac.h, 389
- tropo_zm.c, 571, 574
 - main, 571
 - NT, 571
- ts
 - met_t, 62
- tt
 - met_t, 63
- turb_dx_strat
 - ctl_t, 39
- turb_dx_trop
 - ctl_t, 39
- turb_dz_strat
 - ctl_t, 40
- turb_dz_trop
 - ctl_t, 40
- turb_mesox
 - ctl_t, 40
- turb_mesoz
 - ctl_t, 40
- TVIRT
 - libtrac.h, 313
- u
 - met_t, 65
- us
 - met_t, 62
- uvwp
 - cache_t, 6
- v
 - met_t, 65
- vert_coord_ap
 - ctl_t, 21
- vert_coord_met
 - ctl_t, 21
- vert_vel
 - ctl_t, 21
- vs
 - met_t, 62
- w
 - met_t, 65
- WARN
 - libtrac.h, 314
- wet_depo_bc_a
 - ctl_t, 47
- wet_depo_bc_b
 - ctl_t, 47
- wet_depo_bc_h
 - ctl_t, 48
- wet_depo_bc_ret_ratio
 - ctl_t, 48
- wet_depo_ic_a
 - ctl_t, 47
- wet_depo_ic_b
 - ctl_t, 48
- wet_depo_ic_h
 - ctl_t, 48
- wet_depo_ic_ret_ratio
 - ctl_t, 48
- wet_depo_pre
 - ctl_t, 47
- wind.c, 577, 579
 - main, 577
- write_atm
 - libtrac.c, 184
 - libtrac.h, 390
- write_atm_asc
 - libtrac.c, 185
 - libtrac.h, 391
- write_atm_bin
 - libtrac.c, 186
 - libtrac.h, 392
- write_atm_clams
 - libtrac.c, 187
 - libtrac.h, 393
- write_atm_nc
 - libtrac.c, 189
 - libtrac.h, 395
- write_csi
 - libtrac.c, 190

- libtrac.h, [396](#)
- write_ens
 - libtrac.c, [193](#)
 - libtrac.h, [399](#)
- write_grid
 - libtrac.c, [194](#)
 - libtrac.h, [400](#)
- write_grid_asc
 - libtrac.c, [197](#)
 - libtrac.h, [403](#)
- write_grid_nc
 - libtrac.c, [198](#)
 - libtrac.h, [404](#)
- write_met
 - libtrac.c, [200](#)
 - libtrac.h, [406](#)
- write_met_bin_2d
 - libtrac.c, [201](#)
 - libtrac.h, [407](#)
- write_met_bin_3d
 - libtrac.c, [202](#)
 - libtrac.h, [408](#)
- write_output
 - trac.c, [521](#)
- write_prof
 - libtrac.c, [203](#)
 - libtrac.h, [409](#)
- write_sample
 - libtrac.c, [206](#)
 - libtrac.h, [412](#)
- write_station
 - libtrac.c, [208](#)
 - libtrac.h, [414](#)
- Z
 - libtrac.h, [313](#)
- z
 - met_t, [64](#)
- ZDIFF
 - libtrac.h, [313](#)
- ZETA
 - libtrac.h, [314](#)
- zeta
 - atm_t, [4](#)
 - met_t, [66](#)
- zeta_dot
 - met_t, [67](#)
- zs
 - met_t, [62](#)
- zt
 - met_t, [63](#)