

MPTRAC

Generated by Doxygen 1.8.11

Contents

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	3
4.1.2	Field Documentation	4
4.2	cache_t Struct Reference	4
4.2.1	Detailed Description	5
4.2.2	Field Documentation	5
4.3	ctl_t Struct Reference	7
4.3.1	Detailed Description	11
4.3.2	Field Documentation	11
4.4	met_t Struct Reference	24
4.4.1	Detailed Description	26
4.4.2	Field Documentation	26

5 File Documentation	28
5.1 atm_conv.c File Reference	28
5.1.1 Detailed Description	29
5.1.2 Function Documentation	29
5.2 atm_conv.c	30
5.3 atm_dist.c File Reference	30
5.3.1 Detailed Description	30
5.3.2 Function Documentation	31
5.4 atm_dist.c	35
5.5 atm_init.c File Reference	39
5.5.1 Detailed Description	39
5.5.2 Function Documentation	39
5.6 atm_init.c	41
5.7 atm_select.c File Reference	43
5.7.1 Detailed Description	43
5.7.2 Function Documentation	43
5.8 atm_select.c	45
5.9 atm_split.c File Reference	47
5.9.1 Detailed Description	47
5.9.2 Function Documentation	47
5.10 atm_split.c	49
5.11 atm_stat.c File Reference	51
5.11.1 Detailed Description	52
5.11.2 Function Documentation	52
5.12 atm_stat.c	54
5.13 day2doy.c File Reference	57
5.13.1 Detailed Description	57
5.13.2 Function Documentation	57
5.14 day2doy.c	58
5.15 doy2day.c File Reference	59

5.15.1 Detailed Description	59
5.15.2 Function Documentation	59
5.16 doy2day.c	60
5.17 jsec2time.c File Reference	60
5.17.1 Detailed Description	60
5.17.2 Function Documentation	61
5.18 jsec2time.c	61
5.19 libtrac.c File Reference	62
5.19.1 Detailed Description	64
5.19.2 Function Documentation	64
5.20 libtrac.c	107
5.21 libtrac.h File Reference	158
5.21.1 Detailed Description	160
5.21.2 Function Documentation	160
5.22 libtrac.h	204
5.23 met_map.c File Reference	214
5.23.1 Detailed Description	214
5.23.2 Function Documentation	214
5.24 met_map.c	217
5.25 met_prof.c File Reference	220
5.25.1 Detailed Description	220
5.25.2 Function Documentation	220
5.26 met_prof.c	223
5.27 met_sample.c File Reference	226
5.27.1 Detailed Description	226
5.27.2 Function Documentation	226
5.28 met_sample.c	228
5.29 met_zm.c File Reference	231
5.29.1 Detailed Description	231
5.29.2 Function Documentation	231

5.30 met_zm.c	234
5.31 nvtxmc.h File Reference	237
5.32 nvtxmc.h	237
5.33 time2jsec.c File Reference	238
5.33.1 Detailed Description	238
5.33.2 Function Documentation	238
5.34 time2jsec.c	239
5.35 trac.c File Reference	239
5.35.1 Detailed Description	240
5.35.2 Function Documentation	240
5.35.3 Variable Documentation	262
5.36 trac.c	263
5.37 tropo.c File Reference	281
5.37.1 Detailed Description	281
5.37.2 Function Documentation	281
5.38 tropo.c	286
5.39 tropo_sample.c File Reference	290
5.39.1 Detailed Description	290
5.39.2 Function Documentation	290
5.40 tropo_sample.c	294
Index	299

1 Main Page

Massive-Parallel Trajectory Calculations (MPTRAC) is a Lagrangian particle dispersion model for the troposphere and stratosphere. This reference manual provides information on the algorithms and data structures used in the code. Further information can be found at:

<https://github.com/slcs-jsc/mptrac>

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

atm_t	Atmospheric data	3
cache_t	Cache data	4
ctl_t	Control parameters	7
met_t	Meteorological data	24

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	28
atm_dist.c	Calculate transport deviations of trajectories	30
atm_init.c	Create atmospheric data file with initial air parcel positions	39
atm_select.c	Extract subsets of air parcels from atmospheric data files	43
atm_split.c	Split air parcels into a larger number of parcels	47
atm_stat.c	Calculate air parcel statistics	51
day2doy.c	Convert date to day of year	57
doy2day.c	Convert day of year to date	59
jsec2time.c	Convert Julian seconds to date	60
libtrac.c	MPTRAC library definitions	62
libtrac.h	MPTRAC library declarations	158
met_map.c	Extract map from meteorological data	214
met_prof.c	Extract vertical profile from meteorological data	220

met_sample.c	Sample meteorological data at given geolocations	226
met_zm.c	Extract zonal mean from meteorological data	231
nvtxmc.h		237
time2jsec.c	Convert date to Julian seconds	238
trac.c	Lagrangian particle dispersion model	239
tropo.c	Create tropopause climatology from meteorological data	281
tropo_sample.c	Sample tropopause climatology	290

4 Data Structure Documentation

4.1 atm_t Struct Reference

Atmospheric data.

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

Data Fields

- int [np](#)
Number of air parcels.
- double [time](#) [NP]
Time [s].
- double [p](#) [NP]
Pressure [hPa].
- double [lon](#) [NP]
Longitude [deg].
- double [lat](#) [NP]
Latitude [deg].
- double [q](#) [NQ][NP]
Quantity data (for various, user-defined attributes).

4.1.1 Detailed Description

Atmospheric data.

Definition at line [657](#) of file [libtrac.h](#).

4.1.2 Field Documentation

4.1.2.1 `int atm_t::np`

Number of air parcels.

Definition at line 660 of file [libtrac.h](#).

4.1.2.2 `double atm_t::time[NP]`

Time [s].

Definition at line 663 of file [libtrac.h](#).

4.1.2.3 `double atm_t::p[NP]`

Pressure [hPa].

Definition at line 666 of file [libtrac.h](#).

4.1.2.4 `double atm_t::lon[NP]`

Longitude [deg].

Definition at line 669 of file [libtrac.h](#).

4.1.2.5 `double atm_t::lat[NP]`

Latitude [deg].

Definition at line 672 of file [libtrac.h](#).

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

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

Definition at line 675 of file [libtrac.h](#).

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

- [libtrac.h](#)

4.2 `cache_t` Struct Reference

Cache data.

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


Data Fields

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

4.2.1 Detailed Description

Cache data.

Definition at line 680 of file `libtrac.h`.

4.2.2 Field Documentation

4.2.2.1 float `cache_t::up`[NP]

Zonal wind perturbation [m/s].

Definition at line 683 of file `libtrac.h`.

4.2.2.2 float `cache_t::vp`[NP]

Meridional wind perturbation [m/s].

Definition at line 686 of file `libtrac.h`.

4.2.2.3 float `cache_t::wp`[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 689 of file `libtrac.h`.

4.2.2.4 double cache_t::iso_var[NP]

Isosurface variables.

Definition at line 692 of file [libtrac.h](#).

4.2.2.5 double cache_t::iso_ps[NP]

Isosurface balloon pressure [hPa].

Definition at line 695 of file [libtrac.h](#).

4.2.2.6 double cache_t::iso_ts[NP]

Isosurface balloon time [s].

Definition at line 698 of file [libtrac.h](#).

4.2.2.7 int cache_t::iso_n

Isosurface balloon number of data points.

Definition at line 701 of file [libtrac.h](#).

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

Cache for reference time of wind standard deviations.

Definition at line 704 of file [libtrac.h](#).

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

Cache for zonal wind standard deviations.

Definition at line 707 of file [libtrac.h](#).

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

Cache for meridional wind standard deviations.

Definition at line 710 of file [libtrac.h](#).

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

Cache for vertical velocity standard deviations.

Definition at line 713 of file [libtrac.h](#).

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

- [libtrac.h](#)

4.3 `ctl_t` Struct Reference

Control parameters.

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

Data Fields

- int `nq`
Number of quantities.
- char `qnt_name` [NQ][LEN]
Quantity names.
- char `qnt_unit` [NQ][LEN]
Quantity units.
- char `qnt_format` [NQ][LEN]
Quantity output format.
- int `qnt_ens`
Quantity array index for ensemble IDs.
- int `qnt_m`
Quantity array index for mass.
- int `qnt_rho`
Quantity array index for particle density.
- int `qnt_r`
Quantity array index for particle radius.
- int `qnt_ps`
Quantity array index for surface pressure.
- int `qnt_pt`
Quantity array index for tropopause pressure.
- int `qnt_z`
Quantity array index for geopotential height.
- int `qnt_p`
Quantity array index for pressure.
- int `qnt_t`
Quantity array index for temperature.
- int `qnt_u`
Quantity array index for zonal wind.
- int `qnt_v`
Quantity array index for meridional wind.
- int `qnt_w`
Quantity array index for vertical velocity.
- int `qnt_h2o`
Quantity array index for water vapor vmr.
- int `qnt_o3`
Quantity array index for ozone vmr.
- int `qnt_lwc`
Quantity array index for cloud liquid water content.
- int `qnt_iwc`
Quantity array index for cloud ice water content.
- int `qnt_pc`
Quantity array index for cloud top pressure.

- int [qnt_hno3](#)
Quantity array index for nitric acid vmr.
- int [qnt_oh](#)
Quantity array index for hydroxyl number concentrations.
- int [qnt_rh](#)
Quantity array index for relative humidity.
- int [qnt_theta](#)
Quantity array index for potential temperature.
- int [qnt_vh](#)
Quantity array index for horizontal wind.
- int [qnt_vz](#)
Quantity array index for vertical velocity.
- int [qnt_pv](#)
Quantity array index for potential vorticity.
- int [qnt_tice](#)
Quantity array index for T_{ice} .
- int [qnt_tsts](#)
Quantity array index for T_{STS} .
- int [qnt_tnat](#)
Quantity array index for T_{NAT} .
- int [qnt_stat](#)
Quantity array index for station flag.
- int [direction](#)
Direction flag (1=forward calculation, -1=backward calculation).
- double [t_start](#)
Start time of simulation [s].
- double [t_stop](#)
Stop time of simulation [s].
- double [dt_mod](#)
Time step of simulation [s].
- double [dt_met](#)
Time step of meteorological data [s].
- int [met_dx](#)
Stride for longitudes.
- int [met_dy](#)
Stride for latitudes.
- int [met_dp](#)
Stride for pressure levels.
- int [met_sx](#)
Smoothing for longitudes.
- int [met_sy](#)
Smoothing for latitudes.
- int [met_sp](#)
Smoothing for pressure levels.
- int [met_np](#)
Number of target pressure levels.
- double [met_p](#) [EP]
Target pressure levels [hPa].
- int [met_tropo](#)
Tropopause definition (0=none, 1=clim, 2=cold point, 3=WMO_1st, 4=WMO_2nd).
- char [met_geopot](#) [LEN]

- *Surface geopotential data file.*
- double `met_dt_out`
Time step for sampling of meteo data along trajectories [s].
- char `met_stage` [LEN]
Command to stage meteo data.
- int `isosurf`
Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
- char `balloon` [LEN]
Balloon position filename.
- double `turb_dx_trop`
Horizontal turbulent diffusion coefficient (troposphere) [m²/s].
- double `turb_dx_strat`
Horizontal turbulent diffusion coefficient (stratosphere) [m²/s].
- double `turb_dz_trop`
Vertical turbulent diffusion coefficient (troposphere) [m²/s].
- double `turb_dz_strat`
Vertical turbulent diffusion coefficient (stratosphere) [m²/s].
- double `turb_mesox`
Horizontal scaling factor for mesoscale wind fluctuations.
- double `turb_mesoz`
Vertical scaling factor for mesoscale wind fluctuations.
- char `species` [LEN]
Species.
- double `molmass`
Molar mass [g/mol].
- double `tdec_trop`
Life time of particles (troposphere) [s].
- double `tdec_strat`
Life time of particles (stratosphere) [s].
- double `oh_chem` [4]
Coefficients for OH chemistry (k0, n, kinf, m).
- double `wet_depo` [4]
Coefficients for wet deposition (A, B, H).
- double `psc_h2o`
H2O volume mixing ratio for PSC analysis.
- double `psc_hno3`
HNO3 volume mixing ratio for PSC analysis.
- char `atm_basename` [LEN]
Baseline of atmospheric data files.
- char `atm_gpfile` [LEN]
Gnuplot file for atmospheric data.
- double `atm_dt_out`
Time step for atmospheric data output [s].
- int `atm_filter`
Time filter for atmospheric data output (0=no, 1=yes).
- int `atm_stride`
Particle index stride for atmospheric data files.
- int `atm_type`
Type of atmospheric data files (0=ASCII, 1=binary, 2=netCDF).
- char `csi_basename` [LEN]
Baseline of CSI data files.

- double [csi_dt_out](#)
Time step for CSI data output [s].
- char [csi_obsfile](#) [LEN]
Observation data file for CSI analysis.
- double [csi_obsmin](#)
Minimum observation index to trigger detection.
- double [csi_modmin](#)
Minimum column density to trigger detection [kg/m²].
- int [csi_nz](#)
Number of altitudes of gridded CSI data.
- double [csi_z0](#)
Lower altitude of gridded CSI data [km].
- double [csi_z1](#)
Upper altitude of gridded CSI data [km].
- int [csi_nx](#)
Number of longitudes of gridded CSI data.
- double [csi_lon0](#)
Lower longitude of gridded CSI data [deg].
- double [csi_lon1](#)
Upper longitude of gridded CSI data [deg].
- int [csi_ny](#)
Number of latitudes of gridded CSI data.
- double [csi_lat0](#)
Lower latitude of gridded CSI data [deg].
- double [csi_lat1](#)
Upper latitude of gridded CSI data [deg].
- char [grid_basename](#) [LEN]
Basename of grid data files.
- char [grid_gpfile](#) [LEN]
Gnuplot file for gridded data.
- double [grid_dt_out](#)
Time step for gridded data output [s].
- int [grid_sparse](#)
Sparse output in grid data files (0=no, 1=yes).
- int [grid_nz](#)
Number of altitudes of gridded data.
- double [grid_z0](#)
Lower altitude of gridded data [km].
- double [grid_z1](#)
Upper altitude of gridded data [km].
- int [grid_nx](#)
Number of longitudes of gridded data.
- double [grid_lon0](#)
Lower longitude of gridded data [deg].
- double [grid_lon1](#)
Upper longitude of gridded data [deg].
- int [grid_ny](#)
Number of latitudes of gridded data.
- double [grid_lat0](#)
Lower latitude of gridded data [deg].
- double [grid_lat1](#)

- Upper latitude of gridded data [deg].*
- char `prof_basename` [LEN]
 - Basename for profile output file.*
- char `prof_obsfile` [LEN]
 - Observation data file for profile output.*
- int `prof_nz`
 - Number of altitudes of gridded profile data.*
- double `prof_z0`
 - Lower altitude of gridded profile data [km].*
- double `prof_z1`
 - Upper altitude of gridded profile data [km].*
- int `prof_nx`
 - Number of longitudes of gridded profile data.*
- double `prof_lon0`
 - Lower longitude of gridded profile data [deg].*
- double `prof_lon1`
 - Upper longitude of gridded profile data [deg].*
- int `prof_ny`
 - Number of latitudes of gridded profile data.*
- double `prof_lat0`
 - Lower latitude of gridded profile data [deg].*
- double `prof_lat1`
 - Upper latitude of gridded profile data [deg].*
- char `ens_basename` [LEN]
 - Basename of ensemble data file.*
- char `stat_basename` [LEN]
 - Basename of station data file.*
- double `stat_lon`
 - Longitude of station [deg].*
- double `stat_lat`
 - Latitude of station [deg].*
- double `stat_r`
 - Search radius around station [km].*

4.3.1 Detailed Description

Control parameters.

Definition at line 308 of file `libtrac.h`.

4.3.2 Field Documentation

4.3.2.1 int `ctl_t::nq`

Number of quantities.

Definition at line 311 of file `libtrac.h`.

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

Quantity names.

Definition at line 314 of file [libtrac.h](#).

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

Quantity units.

Definition at line 317 of file [libtrac.h](#).

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

Quantity output format.

Definition at line 320 of file [libtrac.h](#).

4.3.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 323 of file [libtrac.h](#).

4.3.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 326 of file [libtrac.h](#).

4.3.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 329 of file [libtrac.h](#).

4.3.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 332 of file [libtrac.h](#).

4.3.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 335 of file [libtrac.h](#).

4.3.2.10 int ctl_t::qnt_pt

Quantity array index for tropopause pressure.

Definition at line 338 of file [libtrac.h](#).

4.3.2.11 `int ctl_t::qnt_z`

Quantity array index for geopotential height.

Definition at line 341 of file [libtrac.h](#).

4.3.2.12 `int ctl_t::qnt_p`

Quantity array index for pressure.

Definition at line 344 of file [libtrac.h](#).

4.3.2.13 `int ctl_t::qnt_t`

Quantity array index for temperature.

Definition at line 347 of file [libtrac.h](#).

4.3.2.14 `int ctl_t::qnt_u`

Quantity array index for zonal wind.

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

4.3.2.15 `int ctl_t::qnt_v`

Quantity array index for meridional wind.

Definition at line 353 of file [libtrac.h](#).

4.3.2.16 `int ctl_t::qnt_w`

Quantity array index for vertical velocity.

Definition at line 356 of file [libtrac.h](#).

4.3.2.17 `int ctl_t::qnt_h2o`

Quantity array index for water vapor vmr.

Definition at line 359 of file [libtrac.h](#).

4.3.2.18 `int ctl_t::qnt_o3`

Quantity array index for ozone vmr.

Definition at line 362 of file [libtrac.h](#).

4.3.2.19 `int ctl_t::qnt_lwc`

Quantity array index for cloud liquid water content.

Definition at line 365 of file [libtrac.h](#).

4.3.2.20 `int ctl_t::qnt_iwc`

Quantity array index for cloud ice water content.

Definition at line 368 of file [libtrac.h](#).

4.3.2.21 `int ctl_t::qnt_pc`

Quantity array index for cloud top pressure.

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

4.3.2.22 `int ctl_t::qnt_hno3`

Quantity array index for nitric acid vmr.

Definition at line 374 of file [libtrac.h](#).

4.3.2.23 `int ctl_t::qnt_oh`

Quantity array index for hydroxyl number concentrations.

Definition at line 377 of file [libtrac.h](#).

4.3.2.24 `int ctl_t::qnt_rh`

Quantity array index for relative humidity.

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

4.3.2.25 `int ctl_t::qnt_theta`

Quantity array index for potential temperature.

Definition at line 383 of file [libtrac.h](#).

4.3.2.26 `int ctl_t::qnt_vh`

Quantity array index for horizontal wind.

Definition at line 386 of file [libtrac.h](#).

4.3.2.27 `int ctl_t::qnt_vz`

Quantity array index for vertical velocity.

Definition at line 389 of file [libtrac.h](#).

4.3.2.28 `int ctl_t::qnt_pv`

Quantity array index for potential vorticity.

Definition at line 392 of file [libtrac.h](#).

4.3.2.29 `int ctl_t::qnt_tice`

Quantity array index for T_ice.

Definition at line 395 of file [libtrac.h](#).

4.3.2.30 `int ctl_t::qnt_tsts`

Quantity array index for T_STS.

Definition at line 398 of file [libtrac.h](#).

4.3.2.31 `int ctl_t::qnt_tnat`

Quantity array index for T_NAT.

Definition at line 401 of file [libtrac.h](#).

4.3.2.32 `int ctl_t::qnt_stat`

Quantity array index for station flag.

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

4.3.2.33 `int ctl_t::direction`

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

Definition at line 407 of file [libtrac.h](#).

4.3.2.34 `double ctl_t::t_start`

Start time of simulation [s].

Definition at line 410 of file [libtrac.h](#).

4.3.2.35 `double ctl_t::t_stop`

Stop time of simulation [s].

Definition at line 413 of file [libtrac.h](#).

4.3.2.36 `double ctl_t::dt_mod`

Time step of simulation [s].

Definition at line 416 of file [libtrac.h](#).

4.3.2.37 `double ctl_t::dt_met`

Time step of meteorological data [s].

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

4.3.2.38 `int ctl_t::met_dx`

Stride for longitudes.

Definition at line 422 of file [libtrac.h](#).

4.3.2.39 `int ctl_t::met_dy`

Stride for latitudes.

Definition at line 425 of file [libtrac.h](#).

4.3.2.40 `int ctl_t::met_dp`

Stride for pressure levels.

Definition at line 428 of file [libtrac.h](#).

4.3.2.41 `int ctl_t::met_sx`

Smoothing for longitudes.

Definition at line 431 of file [libtrac.h](#).

4.3.2.42 `int ctl_t::met_sy`

Smoothing for latitudes.

Definition at line 434 of file [libtrac.h](#).

4.3.2.43 `int ctl_t::met_sp`

Smoothing for pressure levels.

Definition at line 437 of file [libtrac.h](#).

4.3.2.44 `int ctl_t::met_np`

Number of target pressure levels.

Definition at line 440 of file [libtrac.h](#).

4.3.2.45 `double ctl_t::met_p[EP]`

Target pressure levels [hPa].

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

4.3.2.46 `int ctl_t::met_tropo`

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

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

4.3.2.47 `char ctl_t::met_geopot[LEN]`

Surface geopotential data file.

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

4.3.2.48 `double ctl_t::met_dt_out`

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

Definition at line 453 of file [libtrac.h](#).

4.3.2.49 `char ctl_t::met_stage[LEN]`

Command to stage meteo data.

Definition at line 456 of file [libtrac.h](#).

4.3.2.50 `int ctl_t::isosurf`

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

Definition at line 460 of file [libtrac.h](#).

4.3.2.51 `char ctl_t::balloon[LEN]`

Balloon position filename.

Definition at line 463 of file [libtrac.h](#).

4.3.2.52 `double ctl_t::turb_dx_trop`

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

Definition at line 466 of file [libtrac.h](#).

4.3.2.53 `double ctl_t::turb_dx_strat`

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

Definition at line 469 of file [libtrac.h](#).

4.3.2.54 `double ctl_t::turb_dz_trop`

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

Definition at line 472 of file [libtrac.h](#).

4.3.2.55 `double ctl_t::turb_dz_strat`

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

Definition at line 475 of file [libtrac.h](#).

4.3.2.56 double ctl_t::turb_mesox

Horizontal scaling factor for mesoscale wind fluctuations.

Definition at line 478 of file [libtrac.h](#).

4.3.2.57 double ctl_t::turb_mesoz

Vertical scaling factor for mesoscale wind fluctuations.

Definition at line 481 of file [libtrac.h](#).

4.3.2.58 char ctl_t::species[LEN]

Species.

Definition at line 484 of file [libtrac.h](#).

4.3.2.59 double ctl_t::molmass

Molar mass [g/mol].

Definition at line 487 of file [libtrac.h](#).

4.3.2.60 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 490 of file [libtrac.h](#).

4.3.2.61 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

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

4.3.2.62 double ctl_t::oh_chem[4]

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

Definition at line 496 of file [libtrac.h](#).

4.3.2.63 double ctl_t::wet_depo[4]

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

Definition at line 499 of file [libtrac.h](#).

4.3.2.64 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 502 of file [libtrac.h](#).

4.3.2.65 `double ctl_t::psc_hno3`

HNO3 volume mixing ratio for PSC analysis.

Definition at line 505 of file [libtrac.h](#).

4.3.2.66 `char ctl_t::atm_basename[LEN]`

Basename of atmospheric data files.

Definition at line 508 of file [libtrac.h](#).

4.3.2.67 `char ctl_t::atm_gpfile[LEN]`

Gnuplot file for atmospheric data.

Definition at line 511 of file [libtrac.h](#).

4.3.2.68 `double ctl_t::atm_dt_out`

Time step for atmospheric data output [s].

Definition at line 514 of file [libtrac.h](#).

4.3.2.69 `int ctl_t::atm_filter`

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

Definition at line 517 of file [libtrac.h](#).

4.3.2.70 `int ctl_t::atm_stride`

Particle index stride for atmospheric data files.

Definition at line 520 of file [libtrac.h](#).

4.3.2.71 `int ctl_t::atm_type`

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

Definition at line 523 of file [libtrac.h](#).

4.3.2.72 `char ctl_t::csi_basename[LEN]`

Basename of CSI data files.

Definition at line 526 of file [libtrac.h](#).

4.3.2.73 `double ctl_t::csi_dt_out`

Time step for CSI data output [s].

Definition at line 529 of file [libtrac.h](#).

4.3.2.74 `char ctl_t::csi_obsfile[LEN]`

Observation data file for CSI analysis.

Definition at line 532 of file [libtrac.h](#).

4.3.2.75 `double ctl_t::csi_obsmin`

Minimum observation index to trigger detection.

Definition at line 535 of file [libtrac.h](#).

4.3.2.76 `double ctl_t::csi_modmin`

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

Definition at line 538 of file [libtrac.h](#).

4.3.2.77 `int ctl_t::csi_nz`

Number of altitudes of gridded CSI data.

Definition at line 541 of file [libtrac.h](#).

4.3.2.78 `double ctl_t::csi_z0`

Lower altitude of gridded CSI data [km].

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

4.3.2.79 `double ctl_t::csi_z1`

Upper altitude of gridded CSI data [km].

Definition at line 547 of file [libtrac.h](#).

4.3.2.80 `int ctl_t::csi_nx`

Number of longitudes of gridded CSI data.

Definition at line 550 of file [libtrac.h](#).

4.3.2.81 `double ctl_t::csi_lon0`

Lower longitude of gridded CSI data [deg].

Definition at line 553 of file [libtrac.h](#).

4.3.2.82 `double ctl_t::csi_lon1`

Upper longitude of gridded CSI data [deg].

Definition at line 556 of file [libtrac.h](#).

4.3.2.83 `int ctl_t::csi_ny`

Number of latitudes of gridded CSI data.

Definition at line 559 of file [libtrac.h](#).

4.3.2.84 `double ctl_t::csi_lat0`

Lower latitude of gridded CSI data [deg].

Definition at line 562 of file [libtrac.h](#).

4.3.2.85 `double ctl_t::csi_lat1`

Upper latitude of gridded CSI data [deg].

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

4.3.2.86 `char ctl_t::grid_basename[LEN]`

Basename of grid data files.

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

4.3.2.87 `char ctl_t::grid_gfile[LEN]`

Gnuplot file for gridded data.

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

4.3.2.88 `double ctl_t::grid_dt_out`

Time step for gridded data output [s].

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

4.3.2.89 `int ctl_t::grid_sparse`

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

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

4.3.2.90 `int ctl_t::grid_nz`

Number of altitudes of gridded data.

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

4.3.2.91 `double ctl_t::grid_z0`

Lower altitude of gridded data [km].

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

4.3.2.92 double ctl_t::grid_z1

Upper altitude of gridded data [km].

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

4.3.2.93 int ctl_t::grid_nx

Number of longitudes of gridded data.

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

4.3.2.94 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

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

4.3.2.95 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

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

4.3.2.96 int ctl_t::grid_ny

Number of latitudes of gridded data.

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

4.3.2.97 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

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

4.3.2.98 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

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

4.3.2.99 char ctl_t::prof_basename[LEN]

Basename for profile output file.

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

4.3.2.100 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

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

4.3.2.101 `int ctl_t::prof_nz`

Number of altitudes of gridded profile data.

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

4.3.2.102 `double ctl_t::prof_z0`

Lower altitude of gridded profile data [km].

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

4.3.2.103 `double ctl_t::prof_z1`

Upper altitude of gridded profile data [km].

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

4.3.2.104 `int ctl_t::prof_nx`

Number of longitudes of gridded profile data.

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

4.3.2.105 `double ctl_t::prof_lon0`

Lower longitude of gridded profile data [deg].

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

4.3.2.106 `double ctl_t::prof_lon1`

Upper longitude of gridded profile data [deg].

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

4.3.2.107 `int ctl_t::prof_ny`

Number of latitudes of gridded profile data.

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

4.3.2.108 `double ctl_t::prof_lat0`

Lower latitude of gridded profile data [deg].

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

4.3.2.109 `double ctl_t::prof_lat1`

Upper latitude of gridded profile data [deg].

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

4.3.2.110 `char ctl_t::ens_basename[LEN]`

Basename of ensemble data file.

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

4.3.2.111 `char ctl_t::stat_basename[LEN]`

Basename of station data file.

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

4.3.2.112 `double ctl_t::stat_lon`

Longitude of station [deg].

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

4.3.2.113 `double ctl_t::stat_lat`

Latitude of station [deg].

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

4.3.2.114 `double ctl_t::stat_r`

Search radius around station [km].

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

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

- [libtrac.h](#)

4.4 `met_t` Struct Reference

Meteorological data.

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

Data Fields

- double [time](#)
Time [s].
- int [nx](#)
Number of longitudes.
- int [ny](#)
Number of latitudes.
- int [np](#)
Number of pressure levels.
- double [lon](#) [EX]
Longitude [deg].
- double [lat](#) [EY]
Latitude [deg].
- double [p](#) [EP]
Pressure [hPa].
- float [ps](#) [EX][EY]
Surface pressure [hPa].
- float [zs](#) [EX][EY]
Geopotential height at the surface [km].
- float [pt](#) [EX][EY]
Tropopause pressure [hPa].
- float [pc](#) [EX][EY]
Cloud top pressure [hPa].
- float [cl](#) [EX][EY]
Total column cloud water [kg/m²].
- float [z](#) [EX][EY][EP]
Geopotential height at model levels [km].
- float [t](#) [EX][EY][EP]
Temperature [K].
- float [u](#) [EX][EY][EP]
Zonal wind [m/s].
- float [v](#) [EX][EY][EP]
Meridional wind [m/s].
- float [w](#) [EX][EY][EP]
Vertical wind [hPa/s].
- float [pv](#) [EX][EY][EP]
Potential vorticity [PVU].
- float [h2o](#) [EX][EY][EP]
Water vapor volume mixing ratio [1].
- float [o3](#) [EX][EY][EP]
Ozone volume mixing ratio [1].
- float [lwc](#) [EX][EY][EP]
Cloud liquid water content [kg/kg].
- float [iwc](#) [EX][EY][EP]
Cloud ice water content [kg/kg].
- float [pl](#) [EX][EY][EP]
Pressure on model levels [hPa].

4.4.1 Detailed Description

Meteorological data.

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

4.4.2 Field Documentation

4.4.2.1 double met_t::time

Time [s].

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

4.4.2.2 int met_t::nx

Number of longitudes.

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

4.4.2.3 int met_t::ny

Number of latitudes.

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

4.4.2.4 int met_t::np

Number of pressure levels.

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

4.4.2.5 double met_t::lon[EX]

Longitude [deg].

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

4.4.2.6 double met_t::lat[EY]

Latitude [deg].

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

4.4.2.7 double met_t::p[EP]

Pressure [hPa].

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

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

Surface pressure [hPa].

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

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

Geopotential height at the surface [km].

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

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

Tropopause pressure [hPa].

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

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

Cloud top pressure [hPa].

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

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

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

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

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

Geopotential height at model levels [km].

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

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

Temperature [K].

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

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

Zonal wind [m/s].

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

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

Meridional wind [m/s].

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

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

Vertical wind [hPa/s].

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

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

Potential vorticity [PVU].

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

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

Water vapor volume mixing ratio [1].

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

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

Ozone volume mixing ratio [1].

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

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

Cloud liquid water content [kg/kg].

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

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

Cloud ice water content [kg/kg].

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

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

Pressure on model levels [hPa].

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

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

- [libtrac.h](#)

5 File Documentation

5.1 atm_conv.c File Reference

Convert file format of air parcel data files.

Functions

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

5.1.1 Detailed Description

Convert file format of air parcel data files.

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

5.1.2 Function Documentation

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

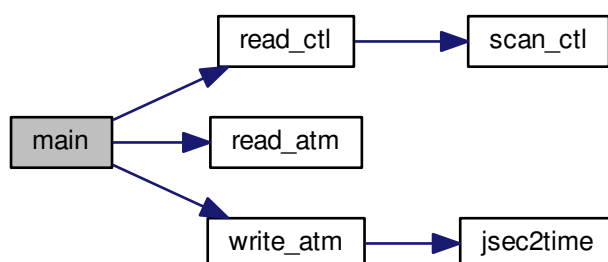
Definition at line 27 of file [atm_conv.c](#).

```

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

```

Here is the call graph for this function:



5.2 atm_conv.c

```

00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm;
00034
00035     /* Check arguments... */
00036     if (argc < 6)
00037         ERRMSG("Give parameters: <ctl> <atm_in> <atm_in_type>"
00038             " <atm_out> <atm_out_type>");
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042
00043     /* Read control parameters... */
00044     read_ctl(argv[1], argc, argv, &ctl);
00045
00046     /* Read atmospheric data... */
00047     ctl.atm_type = atoi(argv[3]);
00048     if (!read_atm(argv[2], &ctl, atm))
00049         ERRMSG("Cannot open file!");
00050
00051     /* Write atmospheric data... */
00052     ctl.atm_type = atoi(argv[5]);
00053     write_atm(argv[4], &ctl, atm, 0);
00054
00055     /* Free... */
00056     free(atm);
00057
00058     return EXIT_SUCCESS;
00059 }

```

5.3 atm_dist.c File Reference

Calculate transport deviations of trajectories.

Functions

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

5.3.1 Detailed Description

Calculate transport deviations of trajectories.

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

5.3.2 Function Documentation

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

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

```

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;
00043
00044     int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00045
00046     /* Allocate... */
00047     ALLOC(atm1, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049     ALLOC(lon1_old, double,
00050         NP);
00051     ALLOC(lat1_old, double,
00052         NP);
00053     ALLOC(z1_old, double,
00054         NP);
00055     ALLOC(lh1, double,
00056         NP);
00057     ALLOC(lv1, double,
00058         NP);
00059     ALLOC(lon2_old, double,
00060         NP);
00061     ALLOC(lat2_old, double,
00062         NP);
00063     ALLOC(z2_old, double,
00064         NP);
00065     ALLOC(lh2, double,
00066         NP);
00067     ALLOC(lv2, double,
00068         NP);
00069     ALLOC(ahtd, double,
00070         NP);
00071     ALLOC(avtd, double,
00072         NP);
00073     ALLOC(aqtd, double,
00074         NP * NQ);
00075     ALLOC(rhtd, double,
00076         NP);
00077     ALLOC(rvtd, double,
00078         NP);
00079     ALLOC(rqtd, double,
00080         NP * NQ);
00081     ALLOC(work, double,
00082         NP);
00083
00084     /* Check arguments... */
00085     if (argc < 6)
00086         ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087             " [<atm2a> <atm2b> ...]");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092     p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093     p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094     lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095     lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096     lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097     lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099     /* Write info... */
00100     printf("Write transport deviations: %s\n", argv[2]);
00101
00102     /* Create output file... */
00103     if (!(out = fopen(argv[2], "w")))
00104         ERRMSG("Cannot create file!");

```

```

00105
00106 /* Write header... */
00107 fprintf(out,
00108     "# $1 = time [s]\n"
00109     "# $2 = time difference [s]\n"
00110     "# $3 = absolute horizontal distance (%s) [km]\n"
00111     "# $4 = relative horizontal distance (%s) [%%]\n"
00112     "# $5 = absolute vertical distance (%s) [km]\n"
00113     "# $6 = relative vertical distance (%s) [%%]\n",
00114     argv[3], argv[3], argv[3], argv[3]);
00115 for (iq = 0; iq < ctl.nq; iq++)
00116     fprintf(out,
00117         "# $d = %s absolute difference (%s) [%s]\n"
00118         "# $d = %s relative difference (%s) [%%]\n",
00119         7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120         8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121 fprintf(out, "# $d = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123 /* Loop over file pairs... */
00124 for (f = 4; f < argc; f += 2) {
00125
00126     /* Read atmospheric data... */
00127     if (!read_atm(argv[f], &ctl, atm1) || !read_atm(argv[f + 1], &ctl, atm2))
00128         continue;
00129
00130     /* Check if structs match... */
00131     if (atm1->np != atm2->np)
00132         ERRMSG("Different numbers of particles!");
00133
00134     /* Get time from filename... */
00135     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00136     year = atoi(tstr);
00137     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00138     mon = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00140     day = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00142     hour = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00144     min = atoi(tstr);
00145     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00146
00147     /* Save initial time... */
00148     if (!init) {
00149         init = 1;
00150         t0 = t;
00151     }
00152
00153     /* Init... */
00154     np = 0;
00155     for (ip = 0; ip < atm1->np; ip++) {
00156         ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00157         for (iq = 0; iq < ctl.nq; iq++)
00158             aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00159     }
00160
00161     /* Loop over air parcels... */
00162     for (ip = 0; ip < atm1->np; ip++) {
00163
00164         /* Check data... */
00165         if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00166             continue;
00167
00168         /* Check ensemble index... */
00169         if (ctl.qnt_ens > 0
00170             && (atm1->q[ctl.qnt_ens][ip] != ens
00171                 || atm2->q[ctl.qnt_ens][ip] != ens))
00172             continue;
00173
00174         /* Check spatial range... */
00175         if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00176             || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00177             || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00178             continue;
00179         if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00180             || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00181             || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00182             continue;
00183
00184         /* Convert coordinates... */
00185         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00186         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00187         z1 = Z(atm1->p[ip]);
00188         z2 = Z(atm2->p[ip]);
00189
00190         /* Calculate absolute transport deviations... */
00191         ahtd[np] = DIST(x1, x2);

```

```

00192     avtd[np] = z1 - z2;
00193     for (iq = 0; iq < ctl.nq; iq++)
00194         aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00195
00196     /* Calculate relative transport deviations... */
00197     if (f > 4) {
00198
00199         /* Get trajectory lengths... */
00200         geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00201         lh1[ip] += DIST(x0, x1);
00202         lv1[ip] += fabs(z1_old[ip] - z1);
00203
00204         geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00205         lh2[ip] += DIST(x0, x2);
00206         lv2[ip] += fabs(z2_old[ip] - z2);
00207
00208         /* Get relative transport deviations... */
00209         if (lh1[ip] + lh2[ip] > 0)
00210             rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00211         if (lv1[ip] + lv2[ip] > 0)
00212             rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00213     }
00214
00215     /* Get relative transport deviations... */
00216     for (iq = 0; iq < ctl.nq; iq++)
00217         rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00218             / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00219
00220     /* Save positions of air parcels... */
00221     lon1_old[ip] = atm1->lon[ip];
00222     lat1_old[ip] = atm1->lat[ip];
00223     z1_old[ip] = z1;
00224
00225     lon2_old[ip] = atm2->lon[ip];
00226     lat2_old[ip] = atm2->lat[ip];
00227     z2_old[ip] = z2;
00228
00229     /* Increment air parcel counter... */
00230     np++;
00231 }
00232
00233 /* Get statistics... */
00234 if (strcmp(argv[3], "mean") == 0) {
00235     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00236     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00237     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00238     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00239     for (iq = 0; iq < ctl.nq; iq++) {
00240         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00241         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00242     }
00243 } else if (strcmp(argv[3], "stddev") == 0) {
00244     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00245     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00246     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00247     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00248     for (iq = 0; iq < ctl.nq; iq++) {
00249         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00250         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00251     }
00252 } else if (strcmp(argv[3], "min") == 0) {
00253     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00254     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00255     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00256     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00257     for (iq = 0; iq < ctl.nq; iq++) {
00258         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00259         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00260     }
00261 } else if (strcmp(argv[3], "max") == 0) {
00262     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00263     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00264     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00265     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00266     for (iq = 0; iq < ctl.nq; iq++) {
00267         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00268         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00269     }
00270 } else if (strcmp(argv[3], "skew") == 0) {
00271     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00272     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00273     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00274     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00275     for (iq = 0; iq < ctl.nq; iq++) {
00276         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00277         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00278     }

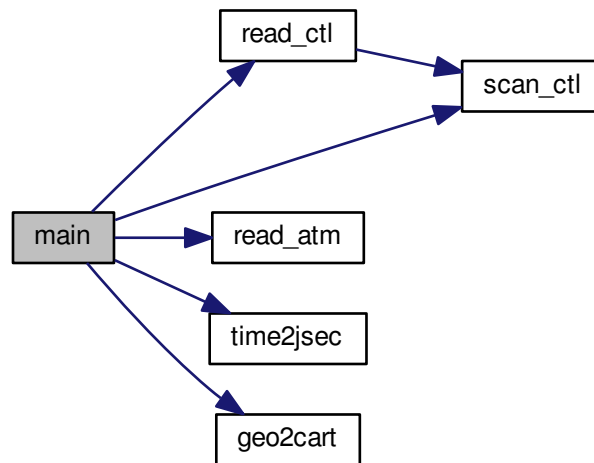
```

```

00279     } else if (strcasecmp(argv[3], "kurt") == 0) {
00280         ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00281         rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00282         avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00283         rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00284         for (iq = 0; iq < ctl.nq; iq++) {
00285             aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00286             rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00287         }
00288     } else if (strcasecmp(argv[3], "median") == 0) {
00289         ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00290         rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00291         avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00292         rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00293         for (iq = 0; iq < ctl.nq; iq++) {
00294             aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00295             rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00296         }
00297     } else if (strcasecmp(argv[3], "absdev") == 0) {
00298         ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
00299         rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
00300         avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
00301         rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
00302         for (iq = 0; iq < ctl.nq; iq++) {
00303             aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
00304             rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
00305         }
00306     } else if (strcasecmp(argv[3], "mad") == 0) {
00307         ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00308         rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00309         avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00310         rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);
00311         for (iq = 0; iq < ctl.nq; iq++) {
00312             aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00313             rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00314         }
00315     } else
00316         ERRMSG("Unknown parameter!");
00317
00318     /* Write output... */
00319     fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00320             ahtdm, rhtdm, avtdm, rvtdm);
00321     for (iq = 0; iq < ctl.nq; iq++) {
00322         fprintf(out, " ");
00323         fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00324         fprintf(out, " ");
00325         fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00326     }
00327     fprintf(out, " %d\n", np);
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Free... */
00334 free(atm1);
00335 free(atm2);
00336 free(lon1_old);
00337 free(lat1_old);
00338 free(z1_old);
00339 free(lh1);
00340 free(lv1);
00341 free(lon2_old);
00342 free(lat2_old);
00343 free(z2_old);
00344 free(lh2);
00345 free(lv2);
00346 free(ahtd);
00347 free(avtd);
00348 free(aqtd);
00349 free(rhtd);
00350 free(rvtd);
00351 free(rqtd);
00352 free(work);
00353
00354 return EXIT_SUCCESS;
00355 }

```

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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     ctl_t ctl;
00032
00033     atm_t *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;
00043
00044     int ens, f, init = 0, ip, iq, np, year, mon, day, hour, min;
00045
00046     /* Allocate... */
00047     ALLOC(atm1, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049     ALLOC(lon1_old, double,

```

```

00050     NP);
00051     ALLOC(lat1_old, double,
00052     NP);
00053     ALLOC(z1_old, double,
00054     NP);
00055     ALLOC(lh1, double,
00056     NP);
00057     ALLOC(lv1, double,
00058     NP);
00059     ALLOC(lon2_old, double,
00060     NP);
00061     ALLOC(lat2_old, double,
00062     NP);
00063     ALLOC(z2_old, double,
00064     NP);
00065     ALLOC(lh2, double,
00066     NP);
00067     ALLOC(lv2, double,
00068     NP);
00069     ALLOC(ahtd, double,
00070     NP);
00071     ALLOC(avtd, double,
00072     NP);
00073     ALLOC(aqtd, double,
00074     NP * NQ);
00075     ALLOC(rhtd, double,
00076     NP);
00077     ALLOC(rvtd, double,
00078     NP);
00079     ALLOC(rqtd, double,
00080     NP * NQ);
00081     ALLOC(work, double,
00082     NP);
00083
00084     /* Check arguments... */
00085     if (argc < 6)
00086         ERRMSG("Give parameters: <ctl> <dist.tab> <param> <atmla> <atmlb>"
00087             " [<atm2a> <atm2b> ...]");
00088
00089     /* Read control parameters... */
00090     read_ctl(argv[1], argc, argv, &ctl);
00091     ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-999", NULL);
00092     p0 = P(scan_ctl(argv[1], argc, argv, "DIST_Z0", -1, "-1000", NULL));
00093     p1 = P(scan_ctl(argv[1], argc, argv, "DIST_Z1", -1, "1000", NULL));
00094     lat0 = scan_ctl(argv[1], argc, argv, "DIST_LAT0", -1, "-1000", NULL);
00095     lat1 = scan_ctl(argv[1], argc, argv, "DIST_LAT1", -1, "1000", NULL);
00096     lon0 = scan_ctl(argv[1], argc, argv, "DIST_LON0", -1, "-1000", NULL);
00097     lon1 = scan_ctl(argv[1], argc, argv, "DIST_LON1", -1, "1000", NULL);
00098
00099     /* Write info... */
00100     printf("Write transport deviations: %s\n", argv[2]);
00101
00102     /* Create output file... */
00103     if (!(out = fopen(argv[2], "w")))
00104         ERRMSG("Cannot create file!");
00105
00106     /* Write header... */
00107     fprintf(out,
00108         "# $1 = time [s]\n"
00109         "# $2 = time difference [s]\n"
00110         "# $3 = absolute horizontal distance (%s) [km]\n"
00111         "# $4 = relative horizontal distance (%s) [%%]\n"
00112         "# $5 = absolute vertical distance (%s) [km]\n"
00113         "# $6 = relative vertical distance (%s) [%%]\n",
00114         argv[3], argv[3], argv[3]);
00115     for (iq = 0; iq < ctl.nq; iq++)
00116         fprintf(out,
00117             "# $d = %s absolute difference (%s) [%s]\n"
00118             "# $d = %s relative difference (%s) [%%]\n",
00119             7 + 2 * iq, ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq],
00120             8 + 2 * iq, ctl.qnt_name[iq], argv[3]);
00121     fprintf(out, "# $d = number of particles\n\n", 7 + 2 * ctl.nq);
00122
00123     /* Loop over file pairs... */
00124     for (f = 4; f < argc; f += 2) {
00125
00126         /* Read atmospheric data... */
00127         if (!read_atm(argv[f], &ctl, atml) || !read_atm(argv[f + 1], &ctl, atm2))
00128             continue;
00129
00130         /* Check if structs match... */
00131         if (atml->np != atm2->np)
00132             ERRMSG("Different numbers of particles!");
00133
00134         /* Get time from filename... */
00135         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00136         year = atoi(tstr);

```



```

00137     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00138     mon = atoi(tstr);
00139     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00140     day = atoi(tstr);
00141     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00142     hour = atoi(tstr);
00143     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00144     min = atoi(tstr);
00145     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00146
00147     /* Save initial time... */
00148     if (!init) {
00149         init = 1;
00150         t0 = t;
00151     }
00152
00153     /* Init... */
00154     np = 0;
00155     for (ip = 0; ip < atm1->np; ip++) {
00156         ahtd[ip] = avtd[ip] = rhtd[ip] = rvtd[ip] = 0;
00157         for (iq = 0; iq < ctl.nq; iq++)
00158             aqtd[iq * NP + ip] = rqtd[iq * NP + ip] = 0;
00159     }
00160
00161     /* Loop over air parcels... */
00162     for (ip = 0; ip < atm1->np; ip++) {
00163
00164         /* Check data... */
00165         if (!gsl_finite(atm1->time[ip]) || !gsl_finite(atm2->time[ip]))
00166             continue;
00167
00168         /* Check ensemble index... */
00169         if (ctl.qnt_ens > 0
00170             && (atm1->q[ctl.qnt_ens][ip] != ens
00171                 || atm2->q[ctl.qnt_ens][ip] != ens))
00172             continue;
00173
00174         /* Check spatial range... */
00175         if (atm1->p[ip] > p0 || atm1->p[ip] < p1
00176             || atm1->lon[ip] < lon0 || atm1->lon[ip] > lon1
00177             || atm1->lat[ip] < lat0 || atm1->lat[ip] > lat1)
00178             continue;
00179         if (atm2->p[ip] > p0 || atm2->p[ip] < p1
00180             || atm2->lon[ip] < lon0 || atm2->lon[ip] > lon1
00181             || atm2->lat[ip] < lat0 || atm2->lat[ip] > lat1)
00182             continue;
00183
00184         /* Convert coordinates... */
00185         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00186         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00187         z1 = Z(atm1->p[ip]);
00188         z2 = Z(atm2->p[ip]);
00189
00190         /* Calculate absolute transport deviations... */
00191         ahtd[np] = DIST(x1, x2);
00192         avtd[np] = z1 - z2;
00193         for (iq = 0; iq < ctl.nq; iq++)
00194             aqtd[iq * NP + np] = atm1->q[iq][ip] - atm2->q[iq][ip];
00195
00196         /* Calculate relative transport deviations... */
00197         if (f > 4) {
00198
00199             /* Get trajectory lengths... */
00200             geo2cart(0, lon1_old[ip], lat1_old[ip], x0);
00201             lh1[ip] += DIST(x0, x1);
00202             lv1[ip] += fabs(z1_old[ip] - z1);
00203
00204             geo2cart(0, lon2_old[ip], lat2_old[ip], x0);
00205             lh2[ip] += DIST(x0, x2);
00206             lv2[ip] += fabs(z2_old[ip] - z2);
00207
00208             /* Get relative transport deviations... */
00209             if (lh1[ip] + lh2[ip] > 0)
00210                 rhtd[np] = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00211             if (lv1[ip] + lv2[ip] > 0)
00212                 rvtd[np] = 200. * (z1 - z2) / (lv1[ip] + lv2[ip]);
00213         }
00214
00215         /* Get relative transport deviations... */
00216         for (iq = 0; iq < ctl.nq; iq++)
00217             rqtd[iq * NP + np] = 200. * (atm1->q[iq][ip] - atm2->q[iq][ip])
00218                 / (fabs(atm1->q[iq][ip]) + fabs(atm2->q[iq][ip]));
00219
00220         /* Save positions of air parcels... */
00221         lon1_old[ip] = atm1->lon[ip];
00222         lat1_old[ip] = atm1->lat[ip];
00223         z1_old[ip] = z1;

```

```

00224
00225     lon2_old[ip] = atm2->lon[ip];
00226     lat2_old[ip] = atm2->lat[ip];
00227     z2_old[ip] = z2;
00228
00229     /* Increment air parcel counter... */
00230     np++;
00231 }
00232
00233 /* Get statistics... */
00234 if (strcasecmp(argv[3], "mean") == 0) {
00235     ahtdm = gsl_stats_mean(ahtd, 1, (size_t) np);
00236     rhtdm = gsl_stats_mean(rhtd, 1, (size_t) np);
00237     avtdm = gsl_stats_mean(avtd, 1, (size_t) np);
00238     rvtdm = gsl_stats_mean(rvtd, 1, (size_t) np);
00239     for (iq = 0; iq < ctl.nq; iq++) {
00240         aqtdm[iq] = gsl_stats_mean(&aqtd[iq * NP], 1, (size_t) np);
00241         rqtdm[iq] = gsl_stats_mean(&rqtd[iq * NP], 1, (size_t) np);
00242     }
00243 } else if (strcasecmp(argv[3], "stddev") == 0) {
00244     ahtdm = gsl_stats_sd(ahtd, 1, (size_t) np);
00245     rhtdm = gsl_stats_sd(rhtd, 1, (size_t) np);
00246     avtdm = gsl_stats_sd(avtd, 1, (size_t) np);
00247     rvtdm = gsl_stats_sd(rvtd, 1, (size_t) np);
00248     for (iq = 0; iq < ctl.nq; iq++) {
00249         aqtdm[iq] = gsl_stats_sd(&aqtd[iq * NP], 1, (size_t) np);
00250         rqtdm[iq] = gsl_stats_sd(&rqtd[iq * NP], 1, (size_t) np);
00251     }
00252 } else if (strcasecmp(argv[3], "min") == 0) {
00253     ahtdm = gsl_stats_min(ahtd, 1, (size_t) np);
00254     rhtdm = gsl_stats_min(rhtd, 1, (size_t) np);
00255     avtdm = gsl_stats_min(avtd, 1, (size_t) np);
00256     rvtdm = gsl_stats_min(rvtd, 1, (size_t) np);
00257     for (iq = 0; iq < ctl.nq; iq++) {
00258         aqtdm[iq] = gsl_stats_min(&aqtd[iq * NP], 1, (size_t) np);
00259         rqtdm[iq] = gsl_stats_min(&rqtd[iq * NP], 1, (size_t) np);
00260     }
00261 } else if (strcasecmp(argv[3], "max") == 0) {
00262     ahtdm = gsl_stats_max(ahtd, 1, (size_t) np);
00263     rhtdm = gsl_stats_max(rhtd, 1, (size_t) np);
00264     avtdm = gsl_stats_max(avtd, 1, (size_t) np);
00265     rvtdm = gsl_stats_max(rvtd, 1, (size_t) np);
00266     for (iq = 0; iq < ctl.nq; iq++) {
00267         aqtdm[iq] = gsl_stats_max(&aqtd[iq * NP], 1, (size_t) np);
00268         rqtdm[iq] = gsl_stats_max(&rqtd[iq * NP], 1, (size_t) np);
00269     }
00270 } else if (strcasecmp(argv[3], "skew") == 0) {
00271     ahtdm = gsl_stats_skew(ahtd, 1, (size_t) np);
00272     rhtdm = gsl_stats_skew(rhtd, 1, (size_t) np);
00273     avtdm = gsl_stats_skew(avtd, 1, (size_t) np);
00274     rvtdm = gsl_stats_skew(rvtd, 1, (size_t) np);
00275     for (iq = 0; iq < ctl.nq; iq++) {
00276         aqtdm[iq] = gsl_stats_skew(&aqtd[iq * NP], 1, (size_t) np);
00277         rqtdm[iq] = gsl_stats_skew(&rqtd[iq * NP], 1, (size_t) np);
00278     }
00279 } else if (strcasecmp(argv[3], "kurt") == 0) {
00280     ahtdm = gsl_stats_kurtosis(ahtd, 1, (size_t) np);
00281     rhtdm = gsl_stats_kurtosis(rhtd, 1, (size_t) np);
00282     avtdm = gsl_stats_kurtosis(avtd, 1, (size_t) np);
00283     rvtdm = gsl_stats_kurtosis(rvtd, 1, (size_t) np);
00284     for (iq = 0; iq < ctl.nq; iq++) {
00285         aqtdm[iq] = gsl_stats_kurtosis(&aqtd[iq * NP], 1, (size_t) np);
00286         rqtdm[iq] = gsl_stats_kurtosis(&rqtd[iq * NP], 1, (size_t) np);
00287     }
00288 } else if (strcasecmp(argv[3], "median") == 0) {
00289     ahtdm = gsl_stats_median(ahtd, 1, (size_t) np);
00290     rhtdm = gsl_stats_median(rhtd, 1, (size_t) np);
00291     avtdm = gsl_stats_median(avtd, 1, (size_t) np);
00292     rvtdm = gsl_stats_median(rvtd, 1, (size_t) np);
00293     for (iq = 0; iq < ctl.nq; iq++) {
00294         aqtdm[iq] = gsl_stats_median(&aqtd[iq * NP], 1, (size_t) np);
00295         rqtdm[iq] = gsl_stats_median(&rqtd[iq * NP], 1, (size_t) np);
00296     }
00297 } else if (strcasecmp(argv[3], "absdev") == 0) {
00298     ahtdm = gsl_stats_absdev(ahtd, 1, (size_t) np);
00299     rhtdm = gsl_stats_absdev(rhtd, 1, (size_t) np);
00300     avtdm = gsl_stats_absdev(avtd, 1, (size_t) np);
00301     rvtdm = gsl_stats_absdev(rvtd, 1, (size_t) np);
00302     for (iq = 0; iq < ctl.nq; iq++) {
00303         aqtdm[iq] = gsl_stats_absdev(&aqtd[iq * NP], 1, (size_t) np);
00304         rqtdm[iq] = gsl_stats_absdev(&rqtd[iq * NP], 1, (size_t) np);
00305     }
00306 } else if (strcasecmp(argv[3], "mad") == 0) {
00307     ahtdm = gsl_stats_mad0(ahtd, 1, (size_t) np, work);
00308     rhtdm = gsl_stats_mad0(rhtd, 1, (size_t) np, work);
00309     avtdm = gsl_stats_mad0(avtd, 1, (size_t) np, work);
00310     rvtdm = gsl_stats_mad0(rvtd, 1, (size_t) np, work);

```

```

00311     for (iq = 0; iq < ctl.nq; iq++) {
00312         aqtdm[iq] = gsl_stats_mad0(&aqtd[iq * NP], 1, (size_t) np, work);
00313         rqtdm[iq] = gsl_stats_mad0(&rqtd[iq * NP], 1, (size_t) np, work);
00314     }
00315 } else
00316     ERRMSG("Unknown parameter!");
00317
00318 /* Write output... */
00319 fprintf(out, "%.2f %.2f %g %g %g %g", t, t - t0,
00320         ahtdm, rhtdm, avtdm, rvtdm);
00321 for (iq = 0; iq < ctl.nq; iq++) {
00322     fprintf(out, " ");
00323     fprintf(out, ctl.qnt_format[iq], aqtdm[iq]);
00324     fprintf(out, " ");
00325     fprintf(out, ctl.qnt_format[iq], rqtdm[iq]);
00326 }
00327 fprintf(out, " %d\n", np);
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Free... */
00334 free(atm1);
00335 free(atm2);
00336 free(lon1_old);
00337 free(lat1_old);
00338 free(z1_old);
00339 free(lh1);
00340 free(lv1);
00341 free(lon2_old);
00342 free(lat2_old);
00343 free(z2_old);
00344 free(lh2);
00345 free(lv2);
00346 free(ahtd);
00347 free(avtd);
00348 free(aqtd);
00349 free(rhtd);
00350 free(rvtd);
00351 free(rqtd);
00352 free(work);
00353
00354 return EXIT_SUCCESS;
00355 }

```

5.5 atm_init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

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

5.5.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

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

5.5.2 Function Documentation

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

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

```

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,
00038         t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
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
00076     /* Initialize random number generator... */
00077     gsl_rng_env_setup();
00078     rng = gsl_rng_alloc(gsl_rng_default);
00079
00080     /* Create grid... */
00081     for (t = t0; t <= t1; t += dt)
00082         for (z = z0; z <= z1; z += dz)
00083             for (lon = lon0; lon <= lon1; lon += dlon)
00084                 for (lat = lat0; lat <= lat1; lat += dlat)
00085                     for (irep = 0; irep < rep; irep++) {
00086
00087                         /* Set position... */
00088                         atm->time[atm->np]
00089                             = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090                                + ut * (gsl_rng_uniform(rng) - 0.5));
00091                         atm->p[atm->np]
00092                             = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093                                + uz * (gsl_rng_uniform(rng) - 0.5));
00094                         atm->lon[atm->np]
00095                             = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                                + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00097                                + ulon * (gsl_rng_uniform(rng) - 0.5));
00098                         do {
00099                             atm->lat[atm->np]
00100                                 = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                                    + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00102                                    + ulat * (gsl_rng_uniform(rng) - 0.5));
00103                         } while (even && gsl_rng_uniform(rng) >
00104                                fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106                         /* Set particle counter... */
00107                         if ((++atm->np) > NP)
00108                             ERRMSG("Too many particles!");
00109                     }
00110
00111     /* Check number of air parcels... */
00112     if (atm->np <= 0)
00113         ERRMSG("Did not create any air parcels!");
00114
00115     /* Initialize mass... */

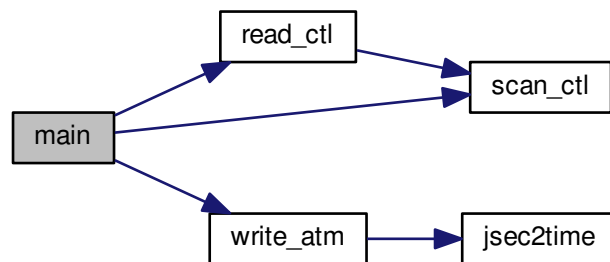
```

```

00116  if (ctl.qnt_m >= 0)
00117      for (ip = 0; ip < atm->np; ip++)
00118          atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120  /* Save data... */
00121  write_atm(argv[2], &ctl, atm, 0);
00122
00123  /* Free... */
00124  gsl_rng_free(rng);
00125  free(atm);
00126
00127  return EXIT_SUCCESS;
00128 }

```

Here is the call graph for this function:



5.6 atm_init.c

```

00001  /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  int main(
00028      int argc,
00029      char *argv[]) {
00030
00031      atm_t *atm;
00032
00033      ctl_t ctl;
00034
00035      gsl_rng *rng;
00036
00037      double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038             t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
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
00076 /* Initialize random number generator... */
00077 gsl_rng_env_setup();
00078 rng = gsl_rng_alloc(gsl_rng_default);
00079
00080 /* Create grid... */
00081 for (t = t0; t <= t1; t += dt)
00082     for (z = z0; z <= z1; z += dz)
00083         for (lon = lon0; lon <= lon1; lon += dlon)
00084             for (lat = lat0; lat <= lat1; lat += dlat)
00085                 for (irep = 0; irep < rep; irep++) {
00086
00087                     /* Set position... */
00088                     atm->time[atm->np]
00089                         = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090                            + ut * (gsl_rng_uniform(rng) - 0.5));
00091                     atm->p[atm->np]
00092                         = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093                            + uz * (gsl_rng_uniform(rng) - 0.5));
00094                     atm->lon[atm->np]
00095                         = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                            + gsl_ran_gaussian_ziggurat(rng, DX2DEG(sx, lat) / 2.3548)
00097                            + ulon * (gsl_rng_uniform(rng) - 0.5));
00098                     do {
00099                         atm->lat[atm->np]
00100                             = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                                + gsl_ran_gaussian_ziggurat(rng, DY2DEG(sx) / 2.3548)
00102                                + ulat * (gsl_rng_uniform(rng) - 0.5));
00103                     } while (even && gsl_rng_uniform(rng) >
00104                             fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106                     /* Set particle counter... */
00107                     if ((++atm->np) > NP)
00108                         ERRMSG("Too many particles!");
00109                 }
00110
00111 /* Check number of air parcels... */
00112 if (atm->np <= 0)
00113     ERRMSG("Did not create any air parcels!");
00114
00115 /* Initialize mass... */
00116 if (ctl.qnt_m >= 0)
00117     for (ip = 0; ip < atm->np; ip++)
00118         atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120 /* Save data... */
00121 write_atm(argv[2], &ctl, atm, 0);
00122
00123 /* Free... */
00124 gsl_rng_free(rng);
00125 free(atm);
00126
00127 return EXIT_SUCCESS;
00128 }

```

5.7 atm_select.c File Reference

Extract subsets of air parcels from atmospheric data files.

Functions

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

5.7.1 Detailed Description

Extract subsets of air parcels from atmospheric data files.

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

5.7.2 Function Documentation

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

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

```

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

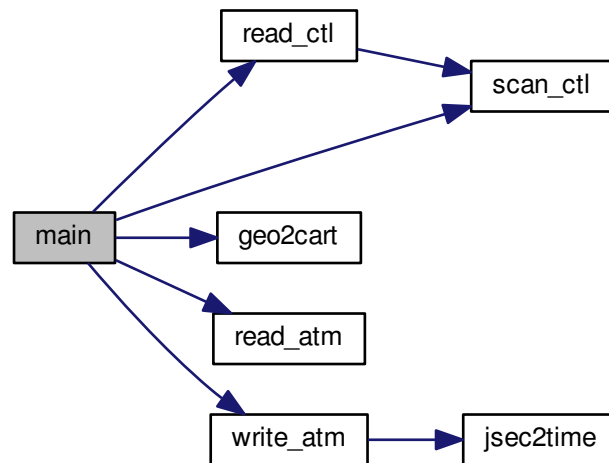
```

```

00077      /* Loop over air parcels... */
00078      for (ip = 0; ip < atm->np; ip += stride) {
00079
00080          /* Check air parcel index... */
00081          if (ip0 != ip1)
00082              if ((ip0 < ip1 && (ip < ip0 || ip > ip1))
00083                  || (ip0 > ip1 && (ip < ip0 && ip > ip1)))
00084              continue;
00085
00086          /* Check time... */
00087          if (t0 != t1)
00088              if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00089                  || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00090              continue;
00091
00092          /* Check vertical distance... */
00093          if (p0 != p1)
00094              if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00095                  || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00096              continue;
00097
00098          /* Check longitude... */
00099          if (lon0 != lon1)
00100              if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00101                  || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00102              continue;
00103
00104          /* Check latitude... */
00105          if (lat0 != lat1)
00106              if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00107                  || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00108              continue;
00109
00110          /* Check horizontal distance... */
00111          if (r0 != r1) {
00112              geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00113              r = DIST(x0, x1);
00114              if ((r1 > r0 && (r < r0 || r > r1))
00115                  || (r1 < r0 && (r < r0 && r > r1)))
00116                  continue;
00117          }
00118
00119          /* Copy data... */
00120          atm2->time[atm2->np] = atm->time[ip];
00121          atm2->p[atm2->np] = atm->p[ip];
00122          atm2->lon[atm2->np] = atm->lon[ip];
00123          atm2->lat[atm2->np] = atm->lat[ip];
00124          for (iq = 0; iq < ctl.nq; iq++)
00125              atm2->q[iq][atm2->np] = atm->q[iq][ip];
00126          if ((++atm2->np) > NP)
00127              ERRMSG("Too many air parcels!");
00128      }
00129  }
00130
00131  /* Close file... */
00132  write_atm(argv[2], &ctl, atm2, 0);
00133
00134  /* Free... */
00135  free(atm);
00136  free(atm2);
00137
00138  return EXIT_SUCCESS;
00139 }

```


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) 2013-2020 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     ctl_t ctl;
00032
00033     atm_t *atm, *atm2;
00034
00035     double lat0, lat1, lon0, lon1, p0, p1, r, r0, r1, rlon, rlat, t0, t1, x0[3],
00036           x1[3];
00037
00038     int f, ip, ip0, ip1, iq, stride;
00039
00040     /* Allocate... */
00041     ALLOC(atm, atm_t, 1);
00042     ALLOC(atm2, atm_t, 1);
00043
00044     /* Check arguments... */
00045     if (argc < 4)
00046         ERRMSG("Give parameters: <ctl> <atm_select> <atm1> [<atm2> ...]");
00047
00048     /* Read control parameters... */
00049     read_ctl(argv[1], argc, argv, &ctl);
  
```

```

00050 stride =
00051     (int) scan_ctl(argv[1], argc, argv, "SELECT_STRIDE", -1, "1", NULL);
00052 ip0 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP0", -1, "0", NULL);
00053 ip1 = (int) scan_ctl(argv[1], argc, argv, "SELECT_IP1", -1, "0", NULL);
00054 t0 = scan_ctl(argv[1], argc, argv, "SELECT_T0", -1, "0", NULL);
00055 t1 = scan_ctl(argv[1], argc, argv, "SELECT_T1", -1, "0", NULL);
00056 p0 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z0", -1, "0", NULL));
00057 p1 = P(scan_ctl(argv[1], argc, argv, "SELECT_Z1", -1, "0", NULL));
00058 lon0 = scan_ctl(argv[1], argc, argv, "SELECT_LON0", -1, "0", NULL);
00059 lon1 = scan_ctl(argv[1], argc, argv, "SELECT_LON1", -1, "0", NULL);
00060 lat0 = scan_ctl(argv[1], argc, argv, "SELECT_LAT0", -1, "0", NULL);
00061 lat1 = scan_ctl(argv[1], argc, argv, "SELECT_LAT1", -1, "0", NULL);
00062 r0 = scan_ctl(argv[1], argc, argv, "SELECT_R0", -1, "0", NULL);
00063 r1 = scan_ctl(argv[1], argc, argv, "SELECT_R1", -1, "0", NULL);
00064 rlon = scan_ctl(argv[1], argc, argv, "SELECT_RLON", -1, "0", NULL);
00065 rlat = scan_ctl(argv[1], argc, argv, "SELECT_RLAT", -1, "0", NULL);
00066
00067 /* Get Cartesian coordinates... */
00068 geo2cart(0, rlon, rlat, x0);
00069
00070 /* Loop over files... */
00071 for (f = 3; f < argc; f++) {
00072
00073     /* Read atmospheric data... */
00074     if (!read_atm(argv[f], &ctl, atm))
00075         continue;
00076
00077     /* Loop over air parcels... */
00078     for (ip = 0; ip < atm->np; ip += stride) {
00079
00080         /* Check air parcel index... */
00081         if (ip0 != ip1)
00082             if ((ip0 < ip1 && (ip < ip0 || ip > ip1))
00083                 || (ip0 > ip1 && (ip < ip0 && ip > ip1)))
00084                 continue;
00085
00086         /* Check time... */
00087         if (t0 != t1)
00088             if ((t1 > t0 && (atm->time[ip] < t0 || atm->time[ip] > t1))
00089                 || (t1 < t0 && (atm->time[ip] < t0 && atm->time[ip] > t1)))
00090                 continue;
00091
00092         /* Check vertical distance... */
00093         if (p0 != p1)
00094             if ((p0 > p1 && (atm->p[ip] > p0 || atm->p[ip] < p1))
00095                 || (p0 < p1 && (atm->p[ip] > p0 && atm->p[ip] < p1)))
00096                 continue;
00097
00098         /* Check longitude... */
00099         if (lon0 != lon1)
00100             if ((lon1 > lon0 && (atm->lon[ip] < lon0 || atm->lon[ip] > lon1))
00101                 || (lon1 < lon0 && (atm->lon[ip] < lon0 && atm->lon[ip] > lon1)))
00102                 continue;
00103
00104         /* Check latitude... */
00105         if (lat0 != lat1)
00106             if ((lat1 > lat0 && (atm->lat[ip] < lat0 || atm->lat[ip] > lat1))
00107                 || (lat1 < lat0 && (atm->lat[ip] < lat0 && atm->lat[ip] > lat1)))
00108                 continue;
00109
00110         /* Check horizontal distance... */
00111         if (r0 != r1) {
00112             geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
00113             r = DIST(x0, x1);
00114             if ((r1 > r0 && (r < r0 || r > r1))
00115                 || (r1 < r0 && (r < r0 && r > r1)))
00116                 continue;
00117         }
00118
00119         /* Copy data... */
00120         atm2->time[atm2->np] = atm->time[ip];
00121         atm2->p[atm2->np] = atm->p[ip];
00122         atm2->lon[atm2->np] = atm->lon[ip];
00123         atm2->lat[atm2->np] = atm->lat[ip];
00124         for (iq = 0; iq < ctl.nq; iq++)
00125             atm2->q[iq][atm2->np] = atm->q[iq][ip];
00126         if ((++atm2->np) > NP)
00127             ERRMSG("Too many air parcels!");
00128     }
00129 }
00130
00131 /* Close file... */
00132 write_atm(argv[2], &ctl, atm2, 0);
00133
00134 /* Free... */
00135 free(atm);
00136 free(atm2);

```

```

00137
00138     return EXIT_SUCCESS;
00139 }

```

5.9 atm_split.c File Reference

Split air parcels into a larger number of parcels.

Functions

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

5.9.1 Detailed Description

Split air parcels into a larger number of parcels.

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

5.9.2 Function Documentation

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

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

```

00029         {
00030
00031     atm_t *atm, *atm2;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     FILE *in;
00038
00039     char kernel[LEN], line[LEN];
00040
00041     double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
00042           t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;
00043
00044     int i, ip, iq, iz, n, nz = 0;
00045
00046     /* Allocate... */
00047     ALLOC(atm, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049
00050     /* Check arguments... */
00051     if (argc < 4)
00052         ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00053
00054     /* Read control parameters... */
00055     read_ctl(argv[1], argc, argv, &ctl);
00056     n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00057     m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00058     dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00059     t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00060     t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00061     dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00062     z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00063     z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00064     dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00065     lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00066     lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00067     lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00068     lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00069     scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);

```

```

00070
00071  /* Init random number generator... */
00072  gsl_rng_env_setup();
00073  rng = gsl_rng_alloc(gsl_rng_default);
00074
00075  /* Read atmospheric data... */
00076  if (!read_atm(argv[2], &ctl, atm))
00077      ERRMSG("Cannot open file!");
00078
00079  /* Read kernel function... */
00080  if (kernel[0] != '-') {
00081
00082      /* Write info... */
00083      printf("Read kernel function: %s\n", kernel);
00084
00085      /* Open file... */
00086      if (!(in = fopen(kernel, "r")))
00087          ERRMSG("Cannot open file!");
00088
00089      /* Read data... */
00090      while (fgets(line, LEN, in))
00091          if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
00092              if ((++nz) >= GZ)
00093                  ERRMSG("Too many height levels!");
00094
00095      /* Close file... */
00096      fclose(in);
00097
00098      /* Normalize kernel function... */
00099      zmax = gsl_stats_max(kz, 1, (size_t) nz);
00100      zmin = gsl_stats_min(kz, 1, (size_t) nz);
00101      kmax = gsl_stats_max(kk, 1, (size_t) nz);
00102      kmin = gsl_stats_min(kk, 1, (size_t) nz);
00103      for (iz = 0; iz < nz; iz++)
00104          kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00105  }
00106
00107  /* Get total and maximum mass... */
00108  if (ctl.qnt_m >= 0)
00109      for (ip = 0; ip < atm->np; ip++) {
00110          mtot += atm->q[ctl.qnt_m][ip];
00111          mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00112      }
00113  if (m > 0)
00114      mtot = m;
00115
00116  /* Loop over air parcels... */
00117  for (i = 0; i < n; i++) {
00118
00119      /* Select air parcel... */
00120      if (ctl.qnt_m >= 0)
00121          do {
00122              ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00123              while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00124          } else
00125              ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00126
00127      /* Set time... */
00128      if (t1 > t0)
00129          atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00130      else
00131          atm2->time[atm2->np] = atm->time[ip]
00132              + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00133
00134      /* Set vertical position... */
00135      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_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00147
00148      /* Set horizontal position... */
00149      if (lon1 > lon0 && lat1 > lat0) {
00150          atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00151          atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00152      } else {
00153          atm2->lon[atm2->np] = atm->lon[ip]
00154              + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00155          atm2->lat[atm2->np] = atm->lat[ip]
00156              + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dy) / 2.3548);

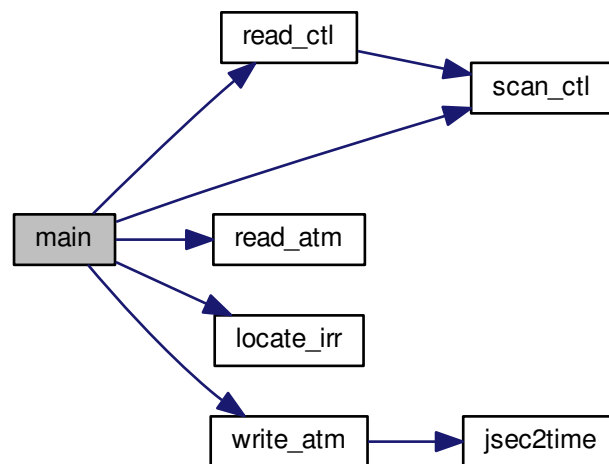
```

```

00157     }
00158
00159     /* Copy quantities... */
00160     for (iq = 0; iq < ctl.nq; iq++)
00161         atm2->q[iq][atm2->np] = atm->q[iq][ip];
00162
00163     /* Adjust mass... */
00164     if (ctl.qnt_m >= 0)
00165         atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00166
00167     /* Increment particle counter... */
00168     if ((++atm2->np) > NP)
00169         ERRMSG("Too many air parcels!");
00170 }
00171
00172 /* Save data and close file... */
00173 write_atm(argv[3], &ctl, atm2, 0);
00174
00175 /* Free... */
00176 free(atm);
00177 free(atm2);
00178
00179 return EXIT_SUCCESS;
00180 }

```

Here is the call graph for this function:



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) 2013–2019 Forschungszentrum Juelich GmbH

```

```

00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     atm_t *atm, *atm2;
00032
00033     ctl_t ctl;
00034
00035     gsl_rng *rng;
00036
00037     FILE *in;
00038
00039     char kernel[LEN], line[LEN];
00040
00041     double dt, dx, dz, k, kk[GZ], kz[GZ], kmin, kmax, m, mmax = 0, mtot = 0,
00042         t0, t1, z, z0, z1, lon0, lon1, lat0, lat1, zmin, zmax;
00043
00044     int i, ip, iq, iz, n, nz = 0;
00045
00046     /* Allocate... */
00047     ALLOC(atm, atm_t, 1);
00048     ALLOC(atm2, atm_t, 1);
00049
00050     /* Check arguments... */
00051     if (argc < 4)
00052         ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00053
00054     /* Read control parameters... */
00055     read_ctl(argv[1], argc, argv, &ctl);
00056     n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00057     m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00058     dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00059     t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00060     t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00061     dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00062     z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00063     z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00064     dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00065     lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00066     lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00067     lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00068     lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00069     scan_ctl(argv[1], argc, argv, "SPLIT_KERNEL", -1, "-", kernel);
00070
00071     /* Init random number generator... */
00072     gsl_rng_env_setup();
00073     rng = gsl_rng_alloc(gsl_rng_default);
00074
00075     /* Read atmospheric data... */
00076     if (!read_atm(argv[2], &ctl, atm))
00077         ERRMSG("Cannot open file!");
00078
00079     /* Read kernel function... */
00080     if (kernel[0] != '-') {
00081
00082         /* Write info... */
00083         printf("Read kernel function: %s\n", kernel);
00084
00085         /* Open file... */
00086         if (!(in = fopen(kernel, "r")))
00087             ERRMSG("Cannot open file!");
00088
00089         /* Read data... */
00090         while (fgets(line, LEN, in))
00091             if (sscanf(line, "%lg %lg", &kz[nz], &kk[nz]) == 2)
00092                 if (++nz) >= GZ
00093                     ERRMSG("Too many height levels!");
00094
00095         /* Close file... */
00096         fclose(in);
00097
00098         /* Normalize kernel function... */
00099         zmax = gsl_stats_max(kz, 1, (size_t) nz);
00100         zmin = gsl_stats_min(kz, 1, (size_t) nz);
00101         kmax = gsl_stats_max(kk, 1, (size_t) nz);
00102         kmin = gsl_stats_min(kk, 1, (size_t) nz);
00103         for (iz = 0; iz < nz; iz++)
00104             kk[iz] = (kk[iz] - kmin) / (kmax - kmin);
00105     }
00106
00107     /* Get total and maximum mass... */
00108     if (ctl.qnt_m >= 0)
00109         for (ip = 0; ip < atm->np; ip++) {

```

```

00110         mtot += atm->q[ctl.qnt_m][ip];
00111         mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00112     }
00113     if (m > 0)
00114         mtot = m;
00115
00116     /* Loop over air parcels... */
00117     for (i = 0; i < n; i++) {
00118
00119         /* Select air parcel... */
00120         if (ctl.qnt_m >= 0)
00121             do {
00122                 ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00123             } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00124         else
00125             ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00126
00127         /* Set time... */
00128         if (t1 > t0)
00129             atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00130         else
00131             atm2->time[atm2->np] = atm->time[ip]
00132                 + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00133
00134         /* Set vertical position... */
00135         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_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00147
00148         /* Set horizontal position... */
00149         if (lon1 > lon0 && lat1 > lat0) {
00150             atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00151             atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00152         } else {
00153             atm2->lon[atm2->np] = atm->lon[ip]
00154                 + gsl_ran_gaussian_ziggurat(rng, DX2DEG(dx, atm->lat[ip]) / 2.3548);
00155             atm2->lat[atm2->np] = atm->lat[ip]
00156                 + gsl_ran_gaussian_ziggurat(rng, DY2DEG(dy, atm->lat[ip]) / 2.3548);
00157         }
00158
00159         /* Copy quantities... */
00160         for (iq = 0; iq < ctl.nq; iq++)
00161             atm2->q[iq][atm2->np] = atm->q[iq][ip];
00162
00163         /* Adjust mass... */
00164         if (ctl.qnt_m >= 0)
00165             atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00166
00167         /* Increment particle counter... */
00168         if ((++atm2->np) > NP)
00169             ERRMSG("Too many air parcels!");
00170     }
00171
00172     /* Save data and close file... */
00173     write_atm(argv[3], &ctl, atm2, 0);
00174
00175     /* Free... */
00176     free(atm);
00177     free(atm2);
00178
00179     return EXIT_SUCCESS;
00180 }

```

5.11 atm_stat.c File Reference

Calculate air parcel statistics.

Functions

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

5.11.1 Detailed Description

Calculate air parcel statistics.

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

5.11.2 Function Documentation

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

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

```

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, 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     printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069     /* Create output file... */
00070     if (!(out = fopen(argv[2], "w")))
00071         ERRMSG("Cannot create file!");
00072
00073     /* Write header... */
00074     fprintf(out,
00075            "# $1 = time [s]\n"
00076            "# $2 = time difference [s]\n"
00077            "# $3 = altitude (%) [km]\n"
00078            "# $4 = longitude (%) [deg]\n"
00079            "# $5 = latitude (%) [deg]\n", argv[3], argv[3], argv[3]);
00080     for (iq = 0; iq < ctl.nq; iq++)
00081         fprintf(out, "# $%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\n", ctl.nq + 6);
00084
00085     /* Loop over files... */
00086     for (f = 4; f < argc; f++) {
00087
00088         /* Read atmospheric data... */
00089         if (!read_atm(argv[f], &ctl, atm))
00090             continue;
00091
00092         /* Get time from filename... */
00093         sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);

```



```

00094     year = atoi(tstr);
00095     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096     mon = atoi(tstr);
00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save intial time... */
00106     if (!init) {
00107         init = 1;
00108         t0 = t;
00109     }
00110
00111     /* Filter data... */
00112     atm_filt->np = 0;
00113     for (ip = 0; ip < atm->np; ip++) {
00114
00115         /* Check time... */
00116         if (!gsl_finite(atm->time[ip]))
00117             continue;
00118
00119         /* Check ensemble index... */
00120         if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00121             continue;
00122
00123         /* Check spatial range... */
00124         if (atm->p[ip] > p0 || atm->p[ip] < p1
00125             || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00126             || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00127             continue;
00128
00129         /* Save data... */
00130         atm_filt->time[atm_filt->np] = atm->time[ip];
00131         atm_filt->p[atm_filt->np] = atm->p[ip];
00132         atm_filt->lon[atm_filt->np] = atm->lon[ip];
00133         atm_filt->lat[atm_filt->np] = atm->lat[ip];
00134         for (iq = 0; iq < ctl.nq; iq++)
00135             atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00136         atm_filt->np++;
00137     }
00138
00139     /* Get heights... */
00140     for (ip = 0; ip < atm_filt->np; ip++)
00141         zs[ip] = Z(atm_filt->p[ip]);
00142
00143     /* Get statistics... */
00144     if (strcmp(argv[3], "mean") == 0) {
00145         zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00146         lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00147         latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00148         for (iq = 0; iq < ctl.nq; iq++)
00149             qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00150     } else if (strcmp(argv[3], "stddev") == 0) {
00151         zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00152         lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00153         latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00154         for (iq = 0; iq < ctl.nq; iq++)
00155             qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcmp(argv[3], "min") == 0) {
00157         zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00158         lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00159         latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00160         for (iq = 0; iq < ctl.nq; iq++)
00161             qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcmp(argv[3], "max") == 0) {
00163         zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00164         lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00165         latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00166         for (iq = 0; iq < ctl.nq; iq++)
00167             qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcmp(argv[3], "skew") == 0) {
00169         zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00170         lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00171         latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00172         for (iq = 0; iq < ctl.nq; iq++)
00173             qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcmp(argv[3], "kurt") == 0) {
00175         zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00176         lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00177         latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00178         for (iq = 0; iq < ctl.nq; iq++)
00179             qm[iq] =
00180                 gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);

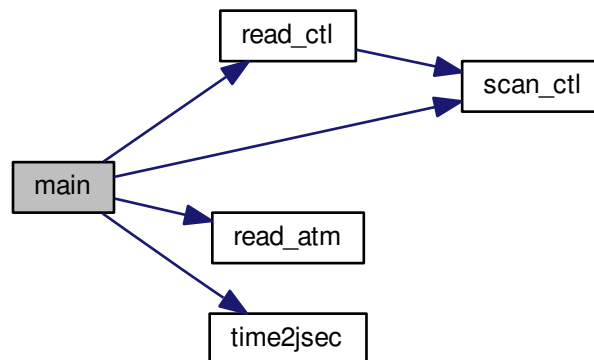
```

```

00181     } else if (strcasecmp(argv[3], "median") == 0) {
00182         zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00183         lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);
00184         latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00185         for (iq = 0; iq < ctl.nq; iq++)
00186             qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187     } else if (strcasecmp(argv[3], "absdev") == 0) {
00188         zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00189         lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00190         latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00191         for (iq = 0; iq < ctl.nq; iq++)
00192             qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193     } else if (strcasecmp(argv[3], "mad") == 0) {
00194         zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00195         lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00196         latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00197         for (iq = 0; iq < ctl.nq; iq++)
00198             qm[iq] =
00199                 gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00200     } else
00201         ERRMSG("Unknown parameter!");
00202
00203     /* Write data... */
00204     fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00205     for (iq = 0; iq < ctl.nq; iq++) {
00206         fprintf(out, " ");
00207         fprintf(out, ctl.qnt_format[iq], qm[iq]);
00208     }
00209     fprintf(out, " %d\n", atm_filt->np);
00210 }
00211
00212 /* Close file... */
00213 fclose(out);
00214
00215 /* Free... */
00216 free(atm);
00217 free(atm_filt);
00218 free(work);
00219 free(zs);
00220
00221 return EXIT_SUCCESS;
00222 }

```

Here is the call graph for this function:



5.12 atm_stat.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify

```

```

00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  int main(
00028      int argc,
00029      char *argv[]) {
00030
00031      ctl_t ctl;
00032
00033      atm_t *atm, *atm_filt;
00034
00035      FILE *out;
00036
00037      char tstr[LEN];
00038
00039      double lat0, lat1, latm, lon0, lon1, lonm, p0, p1,
00040             t, t0, qm[NQ], *work, zm, *zs;
00041
00042      int ens, f, init = 0, ip, iq, year, mon, day, hour, min;
00043
00044      /* Allocate... */
00045      ALLOC(atm, atm_t, 1);
00046      ALLOC(atm_filt, atm_t, 1);
00047      ALLOC(work, double,
00048             NP);
00049      ALLOC(zs, double,
00050             NP);
00051
00052      /* Check arguments... */
00053      if (argc < 4)
00054          ERRMSG("Give parameters: <ctl> <stat.tab> <param> <atm1> [<atm2> ...]");
00055
00056      /* Read control parameters... */
00057      read_ctl(argv[1], argc, argv, &ctl);
00058      ens = (int) scan_ctl(argv[1], argc, argv, "STAT_ENS", -1, "-999", NULL);
00059      p0 = P(scan_ctl(argv[1], argc, argv, "STAT_Z0", -1, "-1000", NULL));
00060      p1 = P(scan_ctl(argv[1], argc, argv, "STAT_Z1", -1, "1000", NULL));
00061      lat0 = scan_ctl(argv[1], argc, argv, "STAT_LAT0", -1, "-1000", NULL);
00062      lat1 = scan_ctl(argv[1], argc, argv, "STAT_LAT1", -1, "1000", NULL);
00063      lon0 = scan_ctl(argv[1], argc, argv, "STAT_LON0", -1, "-1000", NULL);
00064      lon1 = scan_ctl(argv[1], argc, argv, "STAT_LON1", -1, "1000", NULL);
00065
00066      /* Write info... */
00067      printf("Write air parcel statistics: %s\n", argv[2]);
00068
00069      /* Create output file... */
00070      if (!(out = fopen(argv[2], "w")))
00071          ERRMSG("Cannot create file!");
00072
00073      /* Write header... */
00074      fprintf(out,
00075             "# $1 = time [s]\n"
00076             "# $2 = time difference [s]\n"
00077             "# $3 = altitude [%s] [km]\n"
00078             "# $4 = longitude [%s] [deg]\n"
00079             "# $5 = latitude [%s] [deg]\n", argv[3], argv[3], argv[3]);
00080      for (iq = 0; iq < ctl.nq; iq++)
00081          fprintf(out, "# $%d = %s [%s] [%s]\n", iq + 6,
00082                  ctl.qnt_name[iq], argv[3], ctl.qnt_unit[iq]);
00083      fprintf(out, "# $%d = number of particles\n\n", ctl.nq + 6);
00084
00085      /* Loop over files... */
00086      for (f = 4; f < argc; f++) {
00087
00088          /* Read atmospheric data... */
00089          if (!read_atm(argv[f], &ctl, atm))
00090              continue;
00091
00092          /* Get time from filename... */
00093          sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00094          year = atoi(tstr);
00095          sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00096          mon = atoi(tstr);

```

```

00097     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00098     day = atoi(tstr);
00099     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00100     hour = atoi(tstr);
00101     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00102     min = atoi(tstr);
00103     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00104
00105     /* Save intial time... */
00106     if (!init) {
00107         init = 1;
00108         t0 = t;
00109     }
00110
00111     /* Filter data... */
00112     atm_filt->np = 0;
00113     for (ip = 0; ip < atm->np; ip++) {
00114
00115         /* Check time... */
00116         if (!gsl_finite(atm->time[ip]))
00117             continue;
00118
00119         /* Check ensemble index... */
00120         if (ctl.qnt_ens > 0 && atm->q[ctl.qnt_ens][ip] != ens)
00121             continue;
00122
00123         /* Check spatial range... */
00124         if (atm->p[ip] > p0 || atm->p[ip] < p1
00125             || atm->lon[ip] < lon0 || atm->lon[ip] > lon1
00126             || atm->lat[ip] < lat0 || atm->lat[ip] > lat1)
00127             continue;
00128
00129         /* Save data... */
00130         atm_filt->time[atm_filt->np] = atm->time[ip];
00131         atm_filt->p[atm_filt->np] = atm->p[ip];
00132         atm_filt->lon[atm_filt->np] = atm->lon[ip];
00133         atm_filt->lat[atm_filt->np] = atm->lat[ip];
00134         for (iq = 0; iq < ctl.nq; iq++)
00135             atm_filt->q[iq][atm_filt->np] = atm->q[iq][ip];
00136         atm_filt->np++;
00137     }
00138
00139     /* Get heights... */
00140     for (ip = 0; ip < atm_filt->np; ip++)
00141         zs[ip] = Z(atm_filt->p[ip]);
00142
00143     /* Get statistics... */
00144     if (strcasecmp(argv[3], "mean") == 0) {
00145         zm = gsl_stats_mean(zs, 1, (size_t) atm_filt->np);
00146         lonm = gsl_stats_mean(atm_filt->lon, 1, (size_t) atm_filt->np);
00147         latm = gsl_stats_mean(atm_filt->lat, 1, (size_t) atm_filt->np);
00148         for (iq = 0; iq < ctl.nq; iq++)
00149             qm[iq] = gsl_stats_mean(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00150     } else if (strcasecmp(argv[3], "stddev") == 0) {
00151         zm = gsl_stats_sd(zs, 1, (size_t) atm_filt->np);
00152         lonm = gsl_stats_sd(atm_filt->lon, 1, (size_t) atm_filt->np);
00153         latm = gsl_stats_sd(atm_filt->lat, 1, (size_t) atm_filt->np);
00154         for (iq = 0; iq < ctl.nq; iq++)
00155             qm[iq] = gsl_stats_sd(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00156     } else if (strcasecmp(argv[3], "min") == 0) {
00157         zm = gsl_stats_min(zs, 1, (size_t) atm_filt->np);
00158         lonm = gsl_stats_min(atm_filt->lon, 1, (size_t) atm_filt->np);
00159         latm = gsl_stats_min(atm_filt->lat, 1, (size_t) atm_filt->np);
00160         for (iq = 0; iq < ctl.nq; iq++)
00161             qm[iq] = gsl_stats_min(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00162     } else if (strcasecmp(argv[3], "max") == 0) {
00163         zm = gsl_stats_max(zs, 1, (size_t) atm_filt->np);
00164         lonm = gsl_stats_max(atm_filt->lon, 1, (size_t) atm_filt->np);
00165         latm = gsl_stats_max(atm_filt->lat, 1, (size_t) atm_filt->np);
00166         for (iq = 0; iq < ctl.nq; iq++)
00167             qm[iq] = gsl_stats_max(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00168     } else if (strcasecmp(argv[3], "skew") == 0) {
00169         zm = gsl_stats_skew(zs, 1, (size_t) atm_filt->np);
00170         lonm = gsl_stats_skew(atm_filt->lon, 1, (size_t) atm_filt->np);
00171         latm = gsl_stats_skew(atm_filt->lat, 1, (size_t) atm_filt->np);
00172         for (iq = 0; iq < ctl.nq; iq++)
00173             qm[iq] = gsl_stats_skew(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00174     } else if (strcasecmp(argv[3], "kurt") == 0) {
00175         zm = gsl_stats_kurtosis(zs, 1, (size_t) atm_filt->np);
00176         lonm = gsl_stats_kurtosis(atm_filt->lon, 1, (size_t) atm_filt->np);
00177         latm = gsl_stats_kurtosis(atm_filt->lat, 1, (size_t) atm_filt->np);
00178         for (iq = 0; iq < ctl.nq; iq++)
00179             qm[iq] =
00180                 gsl_stats_kurtosis(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00181     } else if (strcasecmp(argv[3], "median") == 0) {
00182         zm = gsl_stats_median(zs, 1, (size_t) atm_filt->np);
00183         lonm = gsl_stats_median(atm_filt->lon, 1, (size_t) atm_filt->np);

```

```

00184     latm = gsl_stats_median(atm_filt->lat, 1, (size_t) atm_filt->np);
00185     for (iq = 0; iq < ctl.nq; iq++)
00186         qm[iq] = gsl_stats_median(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00187 } else if (strcmp(argv[3], "absdev") == 0) {
00188     zm = gsl_stats_absdev(zs, 1, (size_t) atm_filt->np);
00189     lonm = gsl_stats_absdev(atm_filt->lon, 1, (size_t) atm_filt->np);
00190     latm = gsl_stats_absdev(atm_filt->lat, 1, (size_t) atm_filt->np);
00191     for (iq = 0; iq < ctl.nq; iq++)
00192         qm[iq] = gsl_stats_absdev(atm_filt->q[iq], 1, (size_t) atm_filt->np);
00193 } else if (strcmp(argv[3], "mad") == 0) {
00194     zm = gsl_stats_mad0(zs, 1, (size_t) atm_filt->np, work);
00195     lonm = gsl_stats_mad0(atm_filt->lon, 1, (size_t) atm_filt->np, work);
00196     latm = gsl_stats_mad0(atm_filt->lat, 1, (size_t) atm_filt->np, work);
00197     for (iq = 0; iq < ctl.nq; iq++)
00198         qm[iq] =
00199             gsl_stats_mad0(atm_filt->q[iq], 1, (size_t) atm_filt->np, work);
00200 } else
00201     ERRMSG("Unknown parameter!");
00202
00203 /* Write data... */
00204 fprintf(out, "%.2f %.2f %g %g %g", t, t - t0, zm, lonm, latm);
00205 for (iq = 0; iq < ctl.nq; iq++) {
00206     fprintf(out, " ");
00207     fprintf(out, ctl.qnt_format[iq], qm[iq]);
00208 }
00209 fprintf(out, " %d\n", atm_filt->np);
00210 }
00211
00212 /* Close file... */
00213 fclose(out);
00214
00215 /* Free... */
00216 free(atm);
00217 free(atm_filt);
00218 free(work);
00219 free(zs);
00220
00221 return EXIT_SUCCESS;
00222 }

```

5.13 day2doy.c File Reference

Convert date to day of year.

Functions

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

5.13.1 Detailed Description

Convert date to day of year.

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

5.13.2 Function Documentation

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

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

```

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

```

Here is the call graph for this function:



5.14 day2doy.c

```

00001 /*
00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013–2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028     int argc,
00029     char *argv[]) {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 4)
00035         ERRMSG("Give parameters: <year> <mon> <day>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     mon = atoi(argv[2]);
00040     day = atoi(argv[3]);
00041
00042     /* Convert... */
00043     day2doy(year, mon, day, &doy);
00044     printf("%d %d\n", year, doy);
00045
00046     return EXIT_SUCCESS;
00047 }

```

5.15 doy2day.c File Reference

Convert day of year to date.

Functions

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

5.15.1 Detailed Description

Convert day of year to date.

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

5.15.2 Function Documentation

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

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

```
00029         {
00030
00031     int day, doy, mon, year;
00032
00033     /* Check arguments... */
00034     if (argc < 3)
00035         ERRMSG("Give parameters: <year> <doy>");
00036
00037     /* Read arguments... */
00038     year = atoi(argv[1]);
00039     doy = atoi(argv[2]);
00040
00041     /* Convert... */
00042     doy2day(year, doy, &mon, &day);
00043     printf("%d %d %d\n", year, mon, day);
00044
00045     return EXIT_SUCCESS;
00046 }
```

Here is the call graph for this function:



5.16 doy2day.c

```

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

```

5.17 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

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

5.17.1 Detailed Description

Convert Julian seconds to date.

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

5.17.2 Function Documentation

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

Definition at line 27 of file `jsec2time.c`.

```

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

```

Here is the call graph for this function:



5.18 jsec2time.c

```

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

```

```

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

```

5.19 libtrac.c File Reference

MPTRAC library definitions.

Functions

- void [cart2geo](#) (double *x, double *z, double *lon, double *lat)
Convert Cartesian coordinates to geolocation.
- double [clim_hno3](#) (double t, double lat, double p)
Climatology of HNO3 volume mixing ratios.
- double [clim_oh](#) (double t, double lat, double p)
Climatology of OH number concentrations.
- double [clim_tropo](#) (double t, double lat)
Climatology of tropopause pressure.
- void [day2doy](#) (int year, int mon, int day, int *doy)
Get day of year from date.
- void [doy2day](#) (int year, int doy, int *mon, int *day)
Get date from day of year.
- void [geo2cart](#) (double z, double lon, double lat, double *x)
Convert geolocation to Cartesian coordinates.
- void [get_met](#) (ctl_t *ctl, char *metbase, double t, met_t **met0, met_t **met1)
Get meteorological data for given timestep.
- void [get_met_help](#) (double t, int direct, char *metbase, double dt_met, char *filename)
Get meteorological data for timestep.
- void [get_met_replace](#) (char *orig, char *search, char *repl)
Replace template strings in filename.
- void [intpol_met_space_3d](#) (met_t *met, float array[EX][EY][EP], double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteorological data.
- void [intpol_met_space_2d](#) (met_t *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteorological data.
- void [intpol_met_time_3d](#) (met_t *met0, float array0[EX][EY][EP], met_t *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void [intpol_met_time_2d](#) (met_t *met0, float array0[EX][EY], met_t *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void [jsec2time](#) (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)
Convert seconds to date.
- int [locate_irr](#) (double *xx, int n, double x)

- Find array index for irregular grid.*

 - int [locate_reg](#) (double *xx, int n, double x)
- Find array index for regular grid.*

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

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

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

 - void [read_met_cloud](#) ([met_t](#) *met)
- Calculate cloud properties.*

 - void [read_met_extrapolate](#) ([met_t](#) *met)
- Extrapolate meteorological data at lower boundary.*

 - void [read_met_geopot](#) ([met_t](#) *met)
- Calculate geopotential heights.*

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

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

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

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

 - void [read_met_pv](#) ([met_t](#) *met)
- Calculate potential vorticity.*

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

 - void [read_met_surface](#) (int ncid, [met_t](#) *met)
- Read surface data.*

 - void [read_met_tropo](#) ([ctl_t](#) *ctl, [met_t](#) *met)
- Calculate tropopause pressure.*

 - double [scan_ctl](#) (const char *filename, int argc, char *argv[], const char *varname, int arridx, const char *defvalue, char *value)
- Read a control parameter from file or command line.*

 - void [spline](#) (double *x, double *y, int n, double *x2, double *y2, int n2)
- Spline interpolation.*

 - double [stddev](#) (double *data, int n)
- Calculate standard deviation.*

 - void [time2jsec](#) (int year, int mon, int day, int hour, int min, int sec, double remain, double *jsec)
- Convert date to seconds.*

 - void [timer](#) (const char *name, int id, int mode)
- Measure wall-clock time.*

 - void [write_atm](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
- Write atmospheric data.*

 - void [write_csi](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
- Write CSI data.*

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

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

- void [write_prof](#) (const char *filename, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
Write profile data.
- void [write_station](#) (const char *filename, [ctl_t](#) *ctl, [atm_t](#) *atm, double t)
Write station data.

5.19.1 Detailed Description

MPTRAC library definitions.

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

5.19.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

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

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

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

Climatology of HNO3 volume mixing ratios.

Definition at line 304 of file [libtrac.c](#).

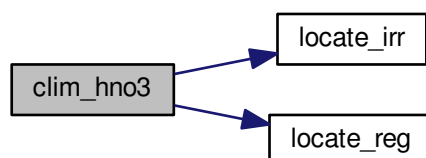
```
00307         {
00308
00309     /* Get seconds since begin of year... */
00310     double sec = FMOD(t, 365.25 * 86400.);
00311     while (sec < 0)
00312         sec += 365.25 * 86400.;
00313
00314     /* Check pressure... */
00315     if (p < clim_hno3_ps[0])
00316         p = clim_hno3_ps[0];
00317     else if (p > clim_hno3_ps[9])
00318         p = clim_hno3_ps[9];
00319
00320     /* Get indices... */
00321     int isec = locate_irr(clim_hno3_secs, 12, sec);
00322     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00323     int ip = locate_irr(clim_hno3_ps, 10, p);
00324
00325     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00326     double aux00 = LIN(clim_hno3_ps[ip],
00327                        clim_hno3_var[isec][ilat][ip],
00328                        clim_hno3_ps[ip + 1],
00329                        clim_hno3_var[isec][ilat][ip + 1], p);
00330     double aux01 = LIN(clim_hno3_ps[ip],
00331                        clim_hno3_var[isec][ilat + 1][ip],
00332                        clim_hno3_ps[ip + 1],
00333                        clim_hno3_var[isec][ilat + 1][ip + 1], p);
00334     double aux10 = LIN(clim_hno3_ps[ip],
00335                        clim_hno3_var[isec + 1][ilat][ip],
00336                        clim_hno3_ps[ip + 1],
00337                        clim_hno3_var[isec + 1][ilat][ip + 1], p);
```

```

00338 double aux11 = LIN(clim_hno3_ps[ip],
00339                   clim_hno3_var[isec + 1][ilat + 1][ip],
00340                   clim_hno3_ps[ip + 1],
00341                   clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00342 aux00 = LIN(clim_hno3_lats[ilat], aux00,
00343            clim_hno3_lats[ilat + 1], aux01, lat);
00344 aux11 = LIN(clim_hno3_lats[ilat], aux10,
00345            clim_hno3_lats[ilat + 1], aux11, lat);
00346 return LIN(clim_hno3_secs[isec], aux00,
00347            clim_hno3_secs[isec + 1], aux11, sec);
00348 }

```

Here is the call graph for this function:



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

Climatology of OH number concentrations.

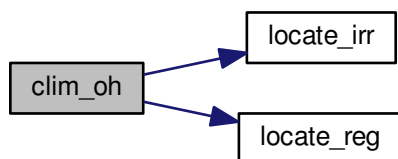
Definition at line 1331 of file [libtrac.c](#).

```

01334 {
01335
01336 /* Get seconds since begin of year... */
01337 double sec = FMOD(t, 365.25 * 86400.);
01338 while (sec < 0)
01339     sec += 365.25 * 86400.;
01340
01341 /* Check pressure... */
01342 if (p < clim_oh_ps[0])
01343     p = clim_oh_ps[0];
01344 else if (p > clim_oh_ps[33])
01345     p = clim_oh_ps[33];
01346
01347 /* Get indices... */
01348 int isec = locate_irr(clim_oh_secs, 12, sec);
01349 int ilat = locate_reg(clim_oh_lats, 18, lat);
01350 int ip = locate_irr(clim_oh_ps, 34, p);
01351
01352 /* Interpolate OH climatology (Pommrich et al., 2014)... */
01353 double aux00 = LIN(clim_oh_ps[ip],
01354                   clim_oh_var[isec][ilat][ip],
01355                   clim_oh_ps[ip + 1],
01356                   clim_oh_var[isec][ilat][ip + 1], p);
01357 double aux01 = LIN(clim_oh_ps[ip],
01358                   clim_oh_var[isec][ilat + 1][ip],
01359                   clim_oh_ps[ip + 1],
01360                   clim_oh_var[isec][ilat + 1][ip + 1], p);
01361 double aux10 = LIN(clim_oh_ps[ip],
01362                   clim_oh_var[isec + 1][ilat][ip],
01363                   clim_oh_ps[ip + 1],
01364                   clim_oh_var[isec + 1][ilat][ip + 1], p);
01365 double aux11 = LIN(clim_oh_ps[ip],
01366                   clim_oh_var[isec + 1][ilat + 1][ip],
01367                   clim_oh_ps[ip + 1],
01368                   clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01369 aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01370 aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01371 return 1e6 * LIN(clim_oh_secs[isec], aux00,
01372                clim_oh_secs[isec + 1], aux11, sec);
01373 }

```

Here is the call graph for this function:



5.19.2.4 double clim_tropo (double t, double lat)

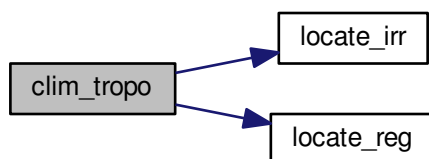
Climatology of tropopause pressure.

Definition at line 1506 of file [libtrac.c](#).

```

01508     {
01509
01510     /* Get seconds since begin of year... */
01511     double sec = FMOD(t, 365.25 * 86400.);
01512     while (sec < 0)
01513         sec += 365.25 * 86400.;
01514
01515     /* Get indices... */
01516     int isec = locate_irr(clim_tropo_secs, 12, sec);
01517     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01518
01519     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01520     double p0 = LIN(clim_tropo_lats[ilat],
01521                    clim_tropo_tps[isec][ilat],
01522                    clim_tropo_lats[ilat + 1],
01523                    clim_tropo_tps[isec][ilat + 1], lat);
01524     double p1 = LIN(clim_tropo_lats[ilat],
01525                    clim_tropo_tps[isec + 1][ilat],
01526                    clim_tropo_lats[ilat + 1],
01527                    clim_tropo_tps[isec + 1][ilat + 1], lat);
01528     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01529 }
  
```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 1533 of file libtrac.c.

```
01537         {
01538
01539     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01540     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01541
01542     /* Get day of year... */
01543     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01544         *doy = d0l[mon - 1] + day - 1;
01545     else
01546         *doy = d0[mon - 1] + day - 1;
01547 }
```

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

Get date from day of year.

Definition at line 1551 of file libtrac.c.

```
01555         {
01556
01557     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01558     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01559     int i;
01560
01561     /* Get month and day... */
01562     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01563         for (i = 11; i >= 0; i--)
01564             if (d0l[i] <= doy)
01565                 break;
01566         *mon = i + 1;
01567         *day = doy - d0l[i] + 1;
01568     } else {
01569         for (i = 11; i >= 0; i--)
01570             if (d0[i] <= doy)
01571                 break;
01572         *mon = i + 1;
01573         *day = doy - d0[i] + 1;
01574     }
01575 }
```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1579 of file libtrac.c.

```
01583         {
01584
01585     double radius = z + RE;
01586     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01587     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01588     x[2] = radius * sin(lat / 180. * M_PI);
01589 }
```

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

Get meteorological data for given timestep.

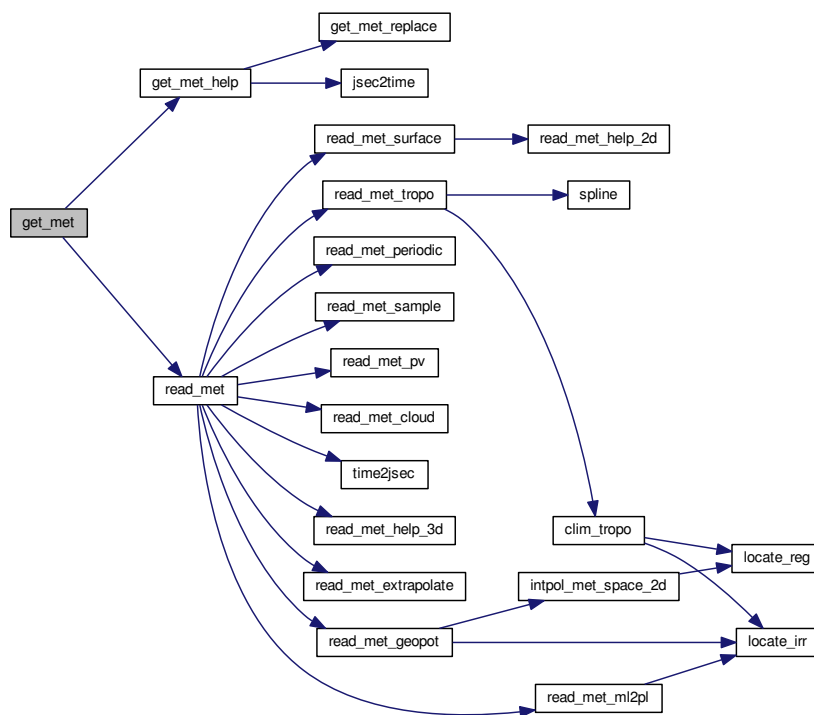
Definition at line 1593 of file libtrac.c.

```

01598         {
01599
01600     static int init, ip, ix, iy;
01601
01602     met_t *mets;
01603
01604     char filename[LEN];
01605
01606     /* Init... */
01607     if (t == ctl->t_start || !init) {
01608         init = 1;
01609
01610         get_met_help(t, -1, metabase, ctl->dt_met, filename);
01611         if (!read_met(ctl, filename, *met0))
01612             ERRMSG("Cannot open file!");
01613
01614         get_met_help(t + 1.0 * ctl->direction, 1, metabase, ctl->
dt_met, filename);
01615         if (!read_met(ctl, filename, *met1))
01616             ERRMSG("Cannot open file!");
01617 #ifdef _OPENACC
01618         met_t *met0up = *met0;
01619         met_t *met1up = *met1;
01620 #pragma acc update device(met0up[:1],met1up[:1])
01621 #endif
01622     }
01623
01624     /* Read new data for forward trajectories... */
01625     if (t > (*met1)->time && ctl->direction == 1) {
01626         mets = *met1;
01627         *met1 = *met0;
01628         *met0 = mets;
01629         get_met_help(t, 1, metabase, ctl->dt_met, filename);
01630         if (!read_met(ctl, filename, *met1))
01631             ERRMSG("Cannot open file!");
01632 #ifdef _OPENACC
01633         met_t *met1up = *met1;
01634 #pragma acc update device(met1up[:1])
01635 #endif
01636     }
01637
01638     /* Read new data for backward trajectories... */
01639     if (t < (*met0)->time && ctl->direction == -1) {
01640         mets = *met1;
01641         *met1 = *met0;
01642         *met0 = mets;
01643         get_met_help(t, -1, metabase, ctl->dt_met, filename);
01644         if (!read_met(ctl, filename, *met0))
01645             ERRMSG("Cannot open file!");
01646 #ifdef _OPENACC
01647         met_t *met0up = *met0;
01648 #pragma acc update device(met0up[:1])
01649 #endif
01650     }
01651
01652     /* Check that grids are consistent... */
01653     if (((*met0)->nx != (*met1)->nx
01654         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01655         ERRMSG("Meteo grid dimensions do not match!");
01656     for (ix = 0; ix < (*met0)->nx; ix++)
01657         if ((*met0)->lon[ix] != (*met1)->lon[ix])
01658             ERRMSG("Meteo grid longitudes do not match!");
01659     for (iy = 0; iy < (*met0)->ny; iy++)
01660         if ((*met0)->lat[iy] != (*met1)->lat[iy])
01661             ERRMSG("Meteo grid latitudes do not match!");
01662     for (ip = 0; ip < (*met0)->np; ip++)
01663         if ((*met0)->p[ip] != (*met1)->p[ip])
01664             ERRMSG("Meteo grid pressure levels do not match!");
01665 }

```


Here is the call graph for this function:



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

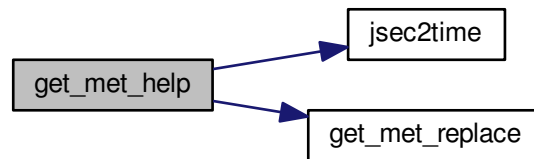
Get meteorological data for timestep.

Definition at line 1669 of file libtrac.c.

```

01674         {
01675
01676     char repl[LEN];
01677
01678     double t6, r;
01679
01680     int year, mon, day, hour, min, sec;
01681
01682     /* Round time to fixed intervals... */
01683     if (direct == -1)
01684         t6 = floor(t / dt_met) * dt_met;
01685     else
01686         t6 = ceil(t / dt_met) * dt_met;
01687
01688     /* Decode time... */
01689     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01690
01691     /* Set filename... */
01692     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01693     sprintf(repl, "%d", year);
01694     get_met_replace(filename, "YYYY", repl);
01695     sprintf(repl, "%02d", mon);
01696     get_met_replace(filename, "MM", repl);
01697     sprintf(repl, "%02d", day);
01698     get_met_replace(filename, "DD", repl);
01699     sprintf(repl, "%02d", hour);
01700     get_met_replace(filename, "HH", repl);
01701 }
  
```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1705 of file libtrac.c.

```

01708         {
01709
01710     char buffer[LEN], *ch;
01711
01712     /* Iterate... */
01713     for (int i = 0; i < 3; i++) {
01714
01715         /* Replace substring... */
01716         if (!(ch = strstr(orig, search)))
01717             return;
01718         strncpy(buffer, orig, (size_t) (ch - orig));
01719         buffer[ch - orig] = 0;
01720         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01721         orig[0] = 0;
01722         strcpy(orig, buffer);
01723     }
01724 }
  
```

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

Spatial interpolation of meteorological data.

Definition at line 1728 of file libtrac.c.

```

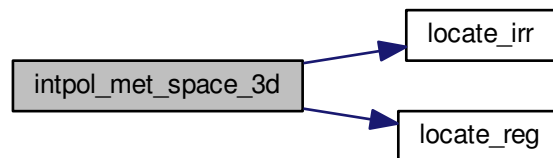
01737         {
01738
01739     /* Check longitude... */
01740     if (met->lon[met->nx - 1] > 180 && lon < 0)
01741         lon += 360;
01742
01743     /* Get interpolation indices and weights... */
01744     if (init) {
01745         ci[0] = locate_irr(met->p, met->np, p);
01746         ci[1] = locate_reg(met->lon, met->nx, lon);
01747         ci[2] = locate_reg(met->lat, met->ny, lat);
01748         cw[0] = (met->p[ci[0] + 1] - p)
01749             / (met->p[ci[0] + 1] - met->p[ci[0]]);
01750         cw[1] = (met->lon[ci[1] + 1] - lon)
01751             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01752         cw[2] = (met->lat[ci[2] + 1] - lat)
01753             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01754     }
01755
01756     /* Interpolate vertically... */
  
```

```

01757 double aux00 =
01758     cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01759     + array[ci[1]][ci[2]][ci[0] + 1];
01760 double aux01 =
01761     cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01762             array[ci[1]][ci[2] + 1][ci[0] + 1])
01763     + array[ci[1]][ci[2] + 1][ci[0] + 1];
01764 double aux10 =
01765     cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01766             array[ci[1] + 1][ci[2]][ci[0] + 1])
01767     + array[ci[1] + 1][ci[2]][ci[0] + 1];
01768 double aux11 =
01769     cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01770             array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01771     + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01772
01773 /* Interpolate horizontally... */
01774 aux00 = cw[2] * (aux00 - aux01) + aux01;
01775 aux11 = cw[2] * (aux10 - aux11) + aux11;
01776 *var = cw[1] * (aux00 - aux11) + aux11;
01777 }

```

Here is the call graph for this function:



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

Spatial interpolation of meteorological data.

Definition at line 1782 of file `libtrac.c`.

```

01790     {
01791
01792     /* Check longitude... */
01793     if (met->lon[met->nx - 1] > 180 && lon < 0)
01794         lon += 360;
01795
01796     /* Get interpolation indices and weights... */
01797     if (init) {
01798         ci[1] = locate_reg(met->lon, met->nx, lon);
01799         ci[2] = locate_reg(met->lat, met->ny, lat);
01800         cw[1] = (met->lon[ci[1] + 1] - lon)
01801             / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01802         cw[2] = (met->lat[ci[2] + 1] - lat)
01803             / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01804     }
01805
01806     /* Set variables... */
01807     double aux00 = array[ci[1]][ci[2]];
01808     double aux01 = array[ci[1]][ci[2] + 1];
01809     double aux10 = array[ci[1] + 1][ci[2]];
01810     double aux11 = array[ci[1] + 1][ci[2] + 1];
01811
01812     /* Interpolate horizontally... */
01813     if (isfinite(aux00) && isfinite(aux01))
01814         aux00 = cw[2] * (aux00 - aux01) + aux01;
01815     else if (cw[2] < 0.5)

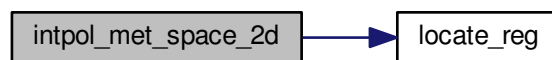
```

```

01816     aux00 = aux01;
01817     if (isfinite(aux10) && isfinite(aux11))
01818         aux11 = cw[2] * (aux10 - aux11) + aux11;
01819     else if (cw[2] > 0.5)
01820         aux11 = aux10;
01821     if (isfinite(aux00) && isfinite(aux11))
01822         *var = cw[1] * (aux00 - aux11) + aux11;
01823     else {
01824         if (cw[1] > 0.5)
01825             *var = aux00;
01826         else
01827             *var = aux11;
01828     }
01829 }

```

Here is the call graph for this function:



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

Temporal interpolation of meteorological data.

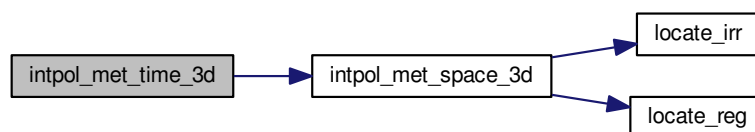
Definition at line 1833 of file `libtrac.c`.

```

01845     {
01846     double var0, var1, wt;
01847     double var0, var1, wt;
01848     /* Spatial interpolation... */
01849     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01850     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
01851     /* Get weighting factor... */
01852     wt = (met1->time - ts) / (met1->time - met0->time);
01853     /* Interpolate... */
01854     *var = wt * (var0 - var1) + var1;
01855     }
01856 }

```

Here is the call graph for this function:



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

Temporal interpolation of meteorological data.

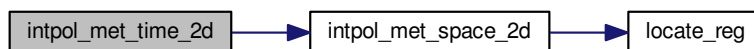
Definition at line 1862 of file `libtrac.c`.

```

01873         {
01874
01875     double var0, var1, wt;
01876
01877     /* Spatial interpolation... */
01878     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01879     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01880
01881     /* Get weighting factor... */
01882     wt = (met1->time - ts) / (met1->time - met0->time);
01883
01884     /* Interpolate... */
01885     *var = wt * (var0 - var1) + var1;
01886 }

```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 1890 of file `libtrac.c`.

```

01898         {
01899
01900     struct tm t0, *t1;
01901
01902     t0.tm_year = 100;
01903     t0.tm_mon = 0;
01904     t0.tm_mday = 1;
01905     t0.tm_hour = 0;
01906     t0.tm_min = 0;
01907     t0.tm_sec = 0;
01908
01909     time_t jsec0 = (time_t) jsec + timegm(&t0);
01910     t1 = gmtime(&jsec0);
01911
01912     *year = t1->tm_year + 1900;
01913     *mon = t1->tm_mon + 1;
01914     *day = t1->tm_mday;
01915     *hour = t1->tm_hour;
01916     *min = t1->tm_min;
01917     *sec = t1->tm_sec;
01918     *remain = jsec - floor(jsec);
01919 }

```

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

Find array index for irregular grid.

Definition at line 1923 of file libtrac.c.

```

01926         {
01927
01928     int ilo = 0;
01929     int ihi = n - 1;
01930     int i = (ihi + ilo) >> 1;
01931
01932     if (xx[i] < xx[i + 1])
01933         while (ihi > ilo + 1) {
01934             i = (ihi + ilo) >> 1;
01935             if (xx[i] > x)
01936                 ihi = i;
01937             else
01938                 ilo = i;
01939         } else
01940             while (ihi > ilo + 1) {
01941                 i = (ihi + ilo) >> 1;
01942                 if (xx[i] <= x)
01943                     ihi = i;
01944                 else
01945                     ilo = i;
01946             }
01947
01948     return ilo;
01949 }
```

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

Find array index for regular grid.

Definition at line 1953 of file libtrac.c.

```

01956         {
01957
01958     /* Calculate index... */
01959     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01960
01961     /* Check range... */
01962     if (i < 0)
01963         i = 0;
01964     else if (i >= n - 2)
01965         i = n - 2;
01966
01967     return i;
01968 }
```

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

Read atmospheric data.

Definition at line 1972 of file libtrac.c.

```

01975         {
01976
01977     FILE *in;
01978
01979     char line[LEN], *tok;
01980
01981     double t0;
01982
01983     int dimid, ip, iq, ncid, varid;
01984
01985     size_t nparts;
01986
01987     /* Init... */
```

```

01988     atm->np = 0;
01989
01990     /* Write info... */
01991     printf("Read atmospheric data: %s\n", filename);
01992
01993     /* Read ASCII data... */
01994     if (ctl->atm_type == 0) {
01995
01996         /* Open file... */
01997         if (!(in = fopen(filename, "r"))) {
01998             WARN("File not found!");
01999             return 0;
02000         }
02001
02002         /* Read line... */
02003         while (fgets(line, LEN, in)) {
02004
02005             /* Read data... */
02006             TOK(line, tok, "%lg", atm->time[atm->np]);
02007             TOK(NULL, tok, "%lg", atm->p[atm->np]);
02008             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02009             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02010             for (iq = 0; iq < ctl->nq; iq++)
02011                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02012
02013             /* Convert altitude to pressure... */
02014             atm->p[atm->np] = P(atm->p[atm->np]);
02015
02016             /* Increment data point counter... */
02017             if (++atm->np > NP)
02018                 ERRMSG("Too many data points!");
02019         }
02020
02021         /* Close file... */
02022         fclose(in);
02023     }
02024
02025     /* Read binary data... */
02026     else if (ctl->atm_type == 1) {
02027
02028         /* Open file... */
02029         if (!(in = fopen(filename, "r")))
02030             return 0;
02031
02032         /* Read data... */
02033         FREAD(&atm->np, int, 1, in);
02034         FREAD(atm->time, double,
02035              (size_t) atm->np,
02036              in);
02037         FREAD(atm->p, double,
02038              (size_t) atm->np,
02039              in);
02040         FREAD(atm->lon, double,
02041              (size_t) atm->np,
02042              in);
02043         FREAD(atm->lat, double,
02044              (size_t) atm->np,
02045              in);
02046         for (iq = 0; iq < ctl->nq; iq++)
02047             FREAD(atm->q[iq], double,
02048                  (size_t) atm->np,
02049                  in);
02050
02051         /* Close file... */
02052         fclose(in);
02053     }
02054
02055     /* Read netCDF data... */
02056     else if (ctl->atm_type == 2) {
02057
02058         /* Open file... */
02059         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02060             return 0;
02061
02062         /* Get dimensions... */
02063         NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02064         NC(nc_inq_dimlen(ncid, dimid, &nparts));
02065         atm->np = (int) nparts;
02066         if (atm->np > NP)
02067             ERRMSG("Too many particles!");
02068
02069         /* Get time... */
02070         NC(nc_inq_varid(ncid, "time", &varid));
02071         NC(nc_get_var_double(ncid, varid, &t0));
02072         for (ip = 0; ip < atm->np; ip++)
02073             atm->time[ip] = t0;
02074

```

```

02075  /* Read geolocations... */
02076  NC(nc_inq_varid(ncid, "PRESS", &varid));
02077  NC(nc_get_var_double(ncid, varid, atm->p));
02078  NC(nc_inq_varid(ncid, "LON", &varid));
02079  NC(nc_get_var_double(ncid, varid, atm->lon));
02080  NC(nc_inq_varid(ncid, "LAT", &varid));
02081  NC(nc_get_var_double(ncid, varid, atm->lat));
02082
02083  /* Read variables... */
02084  if (ctl->qnt_p >= 0)
02085      if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02086          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02087  if (ctl->qnt_t >= 0)
02088      if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02089          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02090  if (ctl->qnt_u >= 0)
02091      if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02092          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02093  if (ctl->qnt_v >= 0)
02094      if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02095          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02096  if (ctl->qnt_w >= 0)
02097      if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02098          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02099  if (ctl->qnt_h2o >= 0)
02100      if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02101          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02102  if (ctl->qnt_o3 >= 0)
02103      if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02104          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02105  if (ctl->qnt_theta >= 0)
02106      if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02107          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02108  if (ctl->qnt_pv >= 0)
02109      if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02110          NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02111
02112  /* Check data... */
02113  for (ip = 0; ip < atm->np; ip++)
02114      if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02115          || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02116          || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02117          || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02118          || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02119      atm->time[ip] = GSL_NAN;
02120      atm->p[ip] = GSL_NAN;
02121      atm->lon[ip] = GSL_NAN;
02122      atm->lat[ip] = GSL_NAN;
02123      for (iq = 0; iq < ctl->nq; iq++)
02124          atm->q[iq][ip] = GSL_NAN;
02125      } else {
02126          if (ctl->qnt_h2o >= 0)
02127              atm->q[ctl->qnt_h2o][ip] *= 1.608;
02128          if (ctl->qnt_pv >= 0)
02129              atm->q[ctl->qnt_pv][ip] *= 1e6;
02130          if (atm->lon[ip] > 180)
02131              atm->lon[ip] -= 360;
02132      }
02133
02134  /* Close file... */
02135  NC(nc_close(ncid));
02136  }
02137
02138  /* Error... */
02139  else
02140      ERRMSG("Atmospheric data type not supported!");
02141
02142  /* Check number of points... */
02143  if (atm->np < 1)
02144      ERRMSG("Can not read any data!");
02145
02146  /* Return success... */
02147  return 1;
02148 }

```

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

Read control parameters.

Definition at line 2152 of file libtrac.c.


```

02156         {
02157
02158     /* Write info... */
02159     printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02160           "(executable: %s | compiled: %s, %s)\n\n",
02161           argv[0], __DATE__, __TIME__);
02162
02163     /* Initialize quantity indices... */
02164     ctl->qnt_ens = -1;
02165     ctl->qnt_m = -1;
02166     ctl->qnt_r = -1;
02167     ctl->qnt_rho = -1;
02168     ctl->qnt_ps = -1;
02169     ctl->qnt_pt = -1;
02170     ctl->qnt_z = -1;
02171     ctl->qnt_p = -1;
02172     ctl->qnt_t = -1;
02173     ctl->qnt_u = -1;
02174     ctl->qnt_v = -1;
02175     ctl->qnt_w = -1;
02176     ctl->qnt_h2o = -1;
02177     ctl->qnt_o3 = -1;
02178     ctl->qnt_lwc = -1;
02179     ctl->qnt_iwc = -1;
02180     ctl->qnt_pc = -1;
02181     ctl->qnt_hno3 = -1;
02182     ctl->qnt_oh = -1;
02183     ctl->qnt_rh = -1;
02184     ctl->qnt_theta = -1;
02185     ctl->qnt_vh = -1;
02186     ctl->qnt_vz = -1;
02187     ctl->qnt_pv = -1;
02188     ctl->qnt_tice = -1;
02189     ctl->qnt_tsts = -1;
02190     ctl->qnt_tnat = -1;
02191     ctl->qnt_stat = -1;
02192
02193     /* Read quantities... */
02194     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02195     if (ctl->nq > NQ)
02196         ERRMSG("Too many quantities!");
02197     for (int iq = 0; iq < ctl->nq; iq++) {
02198
02199         /* Read quantity name and format... */
02200         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02201         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02202               ctl->qnt_format[iq]);
02203
02204         /* Try to identify quantity... */
02205         if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
02206             ctl->qnt_ens = iq;
02207             sprintf(ctl->qnt_unit[iq], "-");
02208         } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
02209             ctl->qnt_m = iq;
02210             sprintf(ctl->qnt_unit[iq], "kg");
02211         } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
02212             ctl->qnt_r = iq;
02213             sprintf(ctl->qnt_unit[iq], "m");
02214         } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
02215             ctl->qnt_rho = iq;
02216             sprintf(ctl->qnt_unit[iq], "kg/m^3");
02217         } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
02218             ctl->qnt_ps = iq;
02219             sprintf(ctl->qnt_unit[iq], "hPa");
02220         } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
02221             ctl->qnt_pt = iq;
02222             sprintf(ctl->qnt_unit[iq], "hPa");
02223         } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
02224             ctl->qnt_z = iq;
02225             sprintf(ctl->qnt_unit[iq], "km");
02226         } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
02227             ctl->qnt_p = iq;
02228             sprintf(ctl->qnt_unit[iq], "hPa");
02229         } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
02230             ctl->qnt_t = iq;
02231             sprintf(ctl->qnt_unit[iq], "K");
02232         } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
02233             ctl->qnt_u = iq;
02234             sprintf(ctl->qnt_unit[iq], "m/s");
02235         } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
02236             ctl->qnt_v = iq;
02237             sprintf(ctl->qnt_unit[iq], "m/s");
02238         } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
02239             ctl->qnt_w = iq;
02240             sprintf(ctl->qnt_unit[iq], "hPa/s");
02241         } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
02242             ctl->qnt_h2o = iq;

```

```

02243     sprintf(ctl->qnt_unit[iq], "ppv");
02244 } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
02245     ctl->qnt_o3 = iq;
02246     sprintf(ctl->qnt_unit[iq], "ppv");
02247 } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
02248     ctl->qnt_lwc = iq;
02249     sprintf(ctl->qnt_unit[iq], "kg/kg");
02250 } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
02251     ctl->qnt_iwc = iq;
02252     sprintf(ctl->qnt_unit[iq], "kg/kg");
02253 } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
02254     ctl->qnt_pc = iq;
02255     sprintf(ctl->qnt_unit[iq], "hPa");
02256 } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
02257     ctl->qnt_hno3 = iq;
02258     sprintf(ctl->qnt_unit[iq], "ppv");
02259 } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
02260     ctl->qnt_oh = iq;
02261     sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02262 } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
02263     ctl->qnt_rh = iq;
02264     sprintf(ctl->qnt_unit[iq], "%");
02265 } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
02266     ctl->qnt_theta = iq;
02267     sprintf(ctl->qnt_unit[iq], "K");
02268 } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
02269     ctl->qnt_vh = iq;
02270     sprintf(ctl->qnt_unit[iq], "m/s");
02271 } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
02272     ctl->qnt_vz = iq;
02273     sprintf(ctl->qnt_unit[iq], "m/s");
02274 } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
02275     ctl->qnt_pv = iq;
02276     sprintf(ctl->qnt_unit[iq], "PVU");
02277 } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
02278     ctl->qnt_tice = iq;
02279     sprintf(ctl->qnt_unit[iq], "K");
02280 } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
02281     ctl->qnt_tsts = iq;
02282     sprintf(ctl->qnt_unit[iq], "K");
02283 } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
02284     ctl->qnt_tnat = iq;
02285     sprintf(ctl->qnt_unit[iq], "K");
02286 } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
02287     ctl->qnt_stat = iq;
02288     sprintf(ctl->qnt_unit[iq], "-");
02289 } else
02290     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02291 }
02292
02293 /* Time steps of simulation... */
02294 ctl->direction =
02295     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02296 if (ctl->direction != -1 && ctl->direction != 1)
02297     ERRMSG("Set DIRECTION to -1 or 1!");
02298 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02299 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
02300
02301 /* Meteorological data... */
02302 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02303 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02304 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02305 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02306 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02307 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02308 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02309 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02310 if (ctl->met_np > EP)
02311     ERRMSG("Too many levels!");
02312 for (int ip = 0; ip < ctl->met_np; ip++)
02313     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02314 ctl->met_tropo =
02315     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02316 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
02317 ctl->met_dt_out =
02318     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02319
02320 /* Isosurface parameters... */
02321 ctl->isosurf =
02322     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02323 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02324
02325 /* Diffusion parameters... */
02326 ctl->turb_dx_trop =
02327     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02328 ctl->turb_dx_strat =
02329     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);

```

```

02330     ctl->turb_dz_trop =
02331         scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02332     ctl->turb_dz_strat =
02333         scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02334     ctl->turb_mesox =
02335         scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02336     ctl->turb_mesoz =
02337         scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02338
02339     /* Species parameters... */
02340     scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02341     if (strcmp(ctl->species, "SO2") == 0) {
02342         ctl->molmass = 64.066;
02343         ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02344         ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02345         ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02346         ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02347         ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
02348         ctl->wet_depo[1] = 0.62; /* (FLEXPART v10.4) */
02349         ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02350         ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02351     } else {
02352         ctl->molmass =
02353             scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02354         ctl->tdec_trop =
02355             scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02356         ctl->tdec_strat =
02357             scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02358         for (int ip = 0; ip < 4; ip++)
02359             ctl->oh_chem[ip] =
02360                 scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02361         for (int ip = 0; ip < 4; ip++)
02362             ctl->wet_depo[ip] =
02363                 scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02364     }
02365
02366     /* PSC analysis... */
02367     ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02368     ctl->psc_hno3 =
02369         scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02370
02371     /* Output of atmospheric data... */
02372     scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
02373     scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02374     ctl->atm_dt_out =
02375         scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02376     ctl->atm_filter =
02377         (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02378     ctl->atm_stride =
02379         (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02380     ctl->atm_type =
02381         (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02382
02383     /* Output of CSI data... */
02384     scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
02385     ctl->csi_dt_out =
02386         scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02387     scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
02388     ctl->csi_obsmin =
02389         scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02390     ctl->csi_modmin =
02391         scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02392     ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02393     ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02394     ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02395     ctl->csi_lon0 =
02396         scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02397     ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02398     ctl->csi_nx =
02399         (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02400     ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02401     ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02402     ctl->csi_ny =
02403         (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02404
02405     /* Output of ensemble data... */
02406     scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
02407
02408     /* Output of grid data... */
02409     scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02410         ctl->grid_basename);
02411     scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);

```

```

02412     ctl->grid_dt_out =
02413         scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02414     ctl->grid_sparse =
02415         (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02416     ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02417     ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02418     ctl->grid_nz =
02419         (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02420     ctl->grid_lon0 =
02421         scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02422     ctl->grid_lon1 =
02423         scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02424     ctl->grid_nx =
02425         (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02426     ctl->grid_lat0 =
02427         scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02428     ctl->grid_lat1 =
02429         scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02430     ctl->grid_ny =
02431         (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02432
02433     /* Output of profile data... */
02434     scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02435         ctl->prof_basename);
02436     scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
02437     prof_obsfile);
02438     ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02439     ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02440     ctl->prof_nz =
02441         (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02442     ctl->prof_lon0 =
02443         scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02444     ctl->prof_lon1 =
02445         scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02446     ctl->prof_nx =
02447         (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02448     ctl->prof_lat0 =
02449         scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02450     ctl->prof_lat1 =
02451         scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02452     ctl->prof_ny =
02453         (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02454
02455     /* Output of station data... */
02456     scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02457         ctl->stat_basename);
02458     ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02459     ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02460     ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02461 }

```

Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 2464 of file libtrac.c.

```

02467     {
02468
02469     char cmd[2 * LEN], levname[LEN], tstr[10];

```

```

02470
02471     int ip, dimid, ncid, varid, year, mon, day, hour;
02472
02473     size_t np, nx, ny;
02474
02475     /* Write info... */
02476     printf("Read meteorological data: %s\n", filename);
02477
02478     /* Get time from filename... */
02479     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
02480     year = atoi(tstr);
02481     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
02482     mon = atoi(tstr);
02483     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
02484     day = atoi(tstr);
02485     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
02486     hour = atoi(tstr);
02487     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
02488
02489     /* Open netCDF file... */
02490     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02491
02492         /* Try to stage meteo file... */
02493         if (ctl->met_stage[0] != '-') {
02494             sprintf(cmd, "%s %d %02d %02d %s", ctl->met_stage,
02495                 year, mon, day, hour, filename);
02496             if (system(cmd) != 0)
02497                 ERRMSG("Error while staging meteo data!");
02498         }
02499
02500         /* Try to open again... */
02501         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02502             WARN("File not found!");
02503             return 0;
02504         }
02505     }
02506
02507     /* Get dimensions... */
02508     NC(nc_inq_dimid(ncid, "lon", &dimid));
02509     NC(nc_inq_dimlen(ncid, dimid, &nx));
02510     if (nx < 2 || nx > EX)
02511         ERRMSG("Number of longitudes out of range!");
02512
02513     NC(nc_inq_dimid(ncid, "lat", &dimid));
02514     NC(nc_inq_dimlen(ncid, dimid, &ny));
02515     if (ny < 2 || ny > EY)
02516         ERRMSG("Number of latitudes out of range!");
02517
02518     sprintf(levname, "lev");
02519     NC(nc_inq_dimid(ncid, levname, &dimid));
02520     NC(nc_inq_dimlen(ncid, dimid, &np));
02521     if (np == 1) {
02522         sprintf(levname, "lev_2");
02523         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
02524             sprintf(levname, "plev");
02525             nc_inq_dimid(ncid, levname, &dimid);
02526         }
02527         NC(nc_inq_dimlen(ncid, dimid, &np));
02528     }
02529     if (np < 2 || np > EP)
02530         ERRMSG("Number of levels out of range!");
02531
02532     /* Store dimensions... */
02533     met->np = (int) np;
02534     met->nx = (int) nx;
02535     met->ny = (int) ny;
02536
02537     /* Get horizontal grid... */
02538     NC(nc_inq_varid(ncid, "lon", &varid));
02539     NC(nc_get_var_double(ncid, varid, met->lon));
02540     NC(nc_inq_varid(ncid, "lat", &varid));
02541     NC(nc_get_var_double(ncid, varid, met->lat));
02542
02543     /* Read meteorological data... */
02544     if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
02545         ERRMSG("Cannot read temperature!");
02546     if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
02547         ERRMSG("Cannot read zonal wind!");
02548     if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
02549         ERRMSG("Cannot read meridional wind!");
02550     if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
02551         WARN("Cannot read vertical velocity!");
02552     if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
02553         WARN("Cannot read specific humidity!");
02554     if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
02555         WARN("Cannot read ozone data!");
02556     if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))

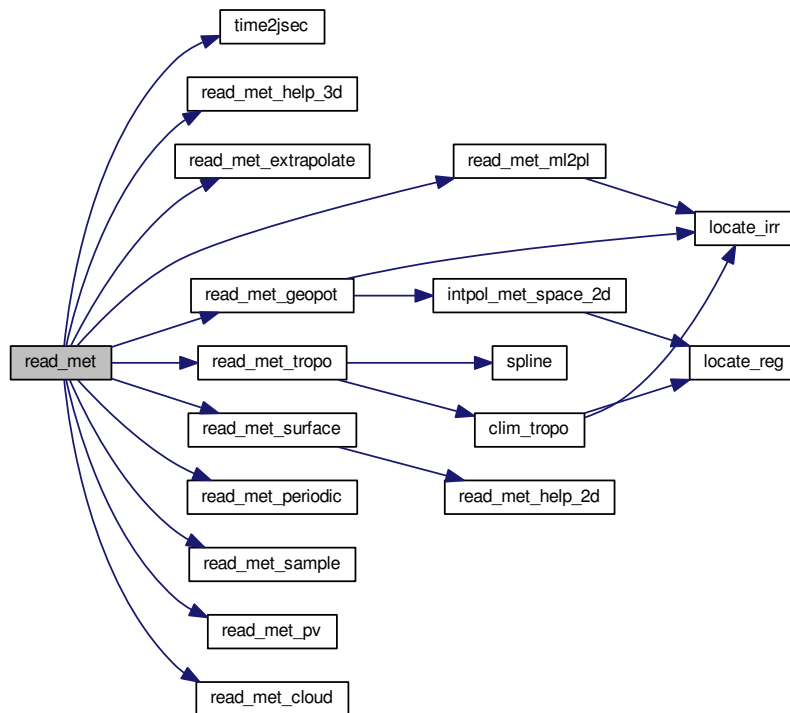
```

```

02557     WARN("Cannot read cloud liquid water content!");
02558     if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
02559         WARN("Cannot read cloud ice water content!");
02560
02561     /* Meteo data on pressure levels... */
02562     if (ctl->met_np <= 0) {
02563
02564         /* Read pressure levels from file... */
02565         NC(nc_inq_varid(ncid, levname, &varid));
02566         NC(nc_get_var_double(ncid, varid, met->p));
02567         for (ip = 0; ip < met->np; ip++)
02568             met->p[ip] /= 100.;
02569
02570         /* Extrapolate data for lower boundary... */
02571         read_met_extrapolate(met);
02572     }
02573
02574     /* Meteo data on model levels... */
02575     else {
02576
02577         /* Read pressure data from file... */
02578         read_met_help_3d(ncid, "pl", "PL", met, met->pl, 0.01f);
02579
02580         /* Interpolate from model levels to pressure levels... */
02581         read_met_ml2pl(ctl, met, met->t);
02582         read_met_ml2pl(ctl, met, met->u);
02583         read_met_ml2pl(ctl, met, met->v);
02584         read_met_ml2pl(ctl, met, met->w);
02585         read_met_ml2pl(ctl, met, met->h2o);
02586         read_met_ml2pl(ctl, met, met->o3);
02587         read_met_ml2pl(ctl, met, met->lwc);
02588         read_met_ml2pl(ctl, met, met->iwc);
02589
02590         /* Set pressure levels... */
02591         met->np = ctl->met_np;
02592         for (ip = 0; ip < met->np; ip++)
02593             met->p[ip] = ctl->met_p[ip];
02594     }
02595
02596     /* Check ordering of pressure levels... */
02597     for (ip = 1; ip < met->np; ip++)
02598         if (met->p[ip - 1] < met->p[ip])
02599             ERRMSG("Pressure levels must be descending!");
02600
02601     /* Read surface data... */
02602     read_met_surface(ncid, met);
02603
02604     /* Create periodic boundary conditions... */
02605     read_met_periodic(met);
02606
02607     /* Downsampling... */
02608     read_met_sample(ctl, met);
02609
02610     /* Calculate geopotential heights... */
02611     read_met_geopot(met);
02612
02613     /* Calculate potential vorticity... */
02614     read_met_pv(met);
02615
02616     /* Calculate tropopause pressure... */
02617     read_met_tropo(ctl, met);
02618
02619     /* Calculate cloud properties... */
02620     read_met_cloud(met);
02621
02622     /* Close file... */
02623     NC(nc_close(ncid));
02624
02625     /* Return success... */
02626     return 1;
02627 }

```

Here is the call graph for this function:



5.19.2.21 void read_met_cloud (met_t * met)

Calculate cloud properties.

Definition at line 2631 of file libtrac.c.

```

02632     {
02633
02634     int ix, iy, ip;
02635
02636     /* Loop over columns... */
02637     #pragma omp parallel for default(shared) private(ix,iy,ip)
02638     for (ix = 0; ix < met->nx; ix++)
02639         for (iy = 0; iy < met->ny; iy++) {
02640
02641         /* Init... */
02642         met->pc[ix][iy] = GSL_NAN;
02643         met->cl[ix][iy] = 0;
02644
02645         /* Loop over pressure levels... */
02646         for (ip = 0; ip < met->np - 1; ip++) {
02647
02648         /* Check pressure... */
02649         if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02650             continue;
02651
02652         /* Get cloud top pressure ... */
02653         if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02654             met->pc[ix][iy] = (float) met->p[ip + 1];
02655
02656         /* Get cloud water... */
02657         met->cl[ix][iy] += (float)
02658             (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02659                 + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02660              * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02661         }
02662     }
02663 }

```

5.19.2.22 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 2667 of file libtrac.c.

```

02668         {
02669
02670     int ip, ip0, ix, iy;
02671
02672     /* Loop over columns... */
02673     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02674     for (ix = 0; ix < met->nx; ix++)
02675         for (iy = 0; iy < met->ny; iy++) {
02676
02677         /* Find lowest valid data point... */
02678         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02679             if (!isfinite(met->t[ix][iy][ip0])
02680                 || !isfinite(met->u[ix][iy][ip0])
02681                 || !isfinite(met->v[ix][iy][ip0])
02682                 || !isfinite(met->w[ix][iy][ip0]))
02683             break;
02684
02685         /* Extrapolate... */
02686         for (ip = ip0; ip >= 0; ip--) {
02687             met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02688             met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02689             met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02690             met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02691             met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02692             met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02693             met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02694             met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02695         }
02696     }
02697 }
```

5.19.2.23 void read_met_geopot (met_t * met)

Calculate geopotential heights.

Definition at line 2701 of file libtrac.c.

```

02702         {
02703
02704     const int dx = 6, dy = 4;
02705
02706     static float help[EX][EY][EP];
02707
02708     double logp[EP], ts, z0, cw[3];
02709
02710     int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];
02711
02712     /* Calculate log pressure... */
02713     for (ip = 0; ip < met->np; ip++)
02714         logp[ip] = log(met->p[ip]);
02715
02716     /* Initialize geopotential heights... */
02717     #pragma omp parallel for default(shared) private(ix,iy,ip)
02718     for (ix = 0; ix < met->nx; ix++)
02719         for (iy = 0; iy < met->ny; iy++)
02720             for (ip = 0; ip < met->np; ip++)
02721                 met->z[ix][iy][ip] = GSL_NAN;
02722
02723     /* Apply hydrostatic equation to calculate geopotential heights... */
02724     #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip,ci,cw)
02725     for (ix = 0; ix < met->nx; ix++)
02726         for (iy = 0; iy < met->ny; iy++) {
02727
02728         /* Get surface height... */
02729         intpol_met_space_2d(met, met->z0, met->lon[ix], met->
02730             lat[iy], &z0, ci,
02731             cw, 1);
02732
02733         /* Find surface pressure level index... */
02734         ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
```

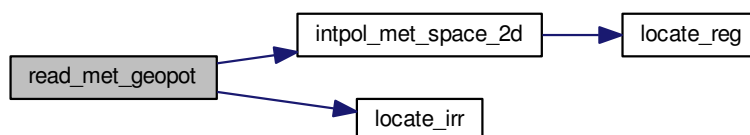


```

02734
02735 /* Get virtual temperature at the surface... */
02736 ts =
02737     LIN(met->p[ip0],
02738         TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
02739         met->p[ip0 + 1],
02740         TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
02741         met->ps[ix][iy]);
02742
02743 /* Upper part of profile... */
02744 met->z[ix][iy][ip0 + 1]
02745     = (float) (z0 + RI / MA / G0 * 0.5
02746               * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
02747                             met->h2o[ix][iy][ip0 + 1]))
02748               * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
02749 for (ip = ip0 + 2; ip < met->np; ip++)
02750     met->z[ix][iy][ip]
02751         = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
02752                   (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
02753                     + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
02754                   * (logp[ip - 1] - logp[ip]));
02755 }
02756
02757 /* Horizontal smoothing... */
02758 #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
02759 for (ix = 0; ix < met->nx; ix++)
02760     for (iy = 0; iy < met->ny; iy++)
02761         for (ip = 0; ip < met->np; ip++) {
02762             n = 0;
02763             help[ix][iy][ip] = 0;
02764             for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
02765                 ix3 = ix2;
02766                 if (ix3 < 0)
02767                     ix3 += met->nx;
02768                 else if (ix3 >= met->nx)
02769                     ix3 -= met->nx;
02770                 for (iy2 = GSL_MAX(iy - dy, 0);
02771                     iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
02772                     if (isfinite(met->z[ix3][iy2][ip])) {
02773                         help[ix][iy][ip] += met->z[ix3][iy2][ip];
02774                         n++;
02775                     }
02776             }
02777             if (n > 0)
02778                 help[ix][iy][ip] /= (float) n;
02779             else
02780                 help[ix][iy][ip] = GSL_NAN;
02781         }
02782
02783 /* Copy data... */
02784 #pragma omp parallel for default(shared) private(ix,iy,ip)
02785 for (ix = 0; ix < met->nx; ix++)
02786     for (iy = 0; iy < met->ny; iy++)
02787         for (ip = 0; ip < met->np; ip++)
02788             met->z[ix][iy][ip] = help[ix][iy][ip];
02789 }

```

Here is the call graph for this function:



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

Read and convert 3D variable from meteorological data file.

Definition at line 2793 of file libtrac.c.

```

02799         {
02800
02801     float *help;
02802
02803     int ip, ix, iy, varid;
02804
02805     /* Check if variable exists... */
02806     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02807         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02808             return 0;
02809
02810     /* Allocate... */
02811     ALLOC(help, float, EX * EY * EP);
02812
02813     /* Read data... */
02814     NC(nc_get_var_float(ncid, varid, help));
02815
02816     /* Copy and check data... */
02817 #pragma omp parallel for default(shared) private(ix,iy,ip)
02818     for (ix = 0; ix < met->nx; ix++)
02819         for (iy = 0; iy < met->ny; iy++)
02820             for (ip = 0; ip < met->np; ip++) {
02821                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
02822                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
02823                     dest[ix][iy][ip] *= scl;
02824                 else
02825                     dest[ix][iy][ip] = GSL_NAN;
02826             }
02827
02828     /* Free... */
02829     free(help);
02830
02831     /* Return... */
02832     return 1;
02833 }

```

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

Read and convert 2D variable from meteorological data file.

Definition at line [2837](#) of file [libtrac.c](#).

```

02843         {
02844
02845     float *help;
02846
02847     int ix, iy, varid;
02848
02849     /* Check if variable exists... */
02850     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02851         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02852             return 0;
02853
02854     /* Allocate... */
02855     ALLOC(help, float, EX * EY);
02856
02857     /* Read data... */
02858     NC(nc_get_var_float(ncid, varid, help));
02859
02860     /* Copy and check data... */
02861 #pragma omp parallel for default(shared) private(ix,iy)
02862     for (ix = 0; ix < met->nx; ix++)
02863         for (iy = 0; iy < met->ny; iy++) {
02864             dest[ix][iy] = help[iy * met->nx + ix];
02865             if (fabsf(dest[ix][iy]) < 1e14f)
02866                 dest[ix][iy] *= scl;
02867             else
02868                 dest[ix][iy] = GSL_NAN;
02869         }
02870
02871     /* Free... */
02872     free(help);
02873
02874     /* Return... */
02875     return 1;
02876 }

```

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

Convert meteorological data from model levels to pressure levels.

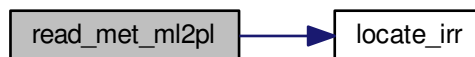
Definition at line 2880 of file [libtrac.c](#).

```

02883         {
02884
02885     double aux[EP], p[EP], pt;
02886
02887     int ip, ip2, ix, iy;
02888
02889     /* Loop over columns... */
02890 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
02891     for (ix = 0; ix < met->nx; ix++)
02892         for (iy = 0; iy < met->ny; iy++) {
02893
02894             /* Copy pressure profile... */
02895             for (ip = 0; ip < met->np; ip++)
02896                 p[ip] = met->pl[ix][iy][ip];
02897
02898             /* Interpolate... */
02899             for (ip = 0; ip < ctl->met_np; ip++) {
02900                 pt = ctl->met_p[ip];
02901                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
02902                     pt = p[0];
02903                 else if ((pt > p[met->np - 1] && p[1] > p[0])
02904                     || (pt < p[met->np - 1] && p[1] < p[0]))
02905                     pt = p[met->np - 1];
02906                 ip2 = locate_irr(p, met->np, pt);
02907                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
02908                     p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
02909             }
02910
02911             /* Copy data... */
02912             for (ip = 0; ip < ctl->met_np; ip++)
02913                 var[ix][iy][ip] = (float) aux[ip];
02914         }
02915 }

```

Here is the call graph for this function:

5.19.2.27 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 2919 of file [libtrac.c](#).

```

02920         {
02921
02922     /* Check longitudes... */
02923     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
02924         + met->lon[1] - met->lon[0] - 360) < 0.01))
02925         return;
02926
02927     /* Increase longitude counter... */
02928     if ((++met->nx) > EX)
02929         ERRMSG("Cannot create periodic boundary conditions!");

```

```

02930
02931  /* Set longitude... */
02932  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
02933
02934  /* Loop over latitudes and pressure levels... */
02935  #pragma omp parallel for default(shared)
02936  for (int iy = 0; iy < met->ny; iy++) {
02937      met->ps[met->nx - 1][iy] = met->ps[0][iy];
02938      met->zs[met->nx - 1][iy] = met->zs[0][iy];
02939      for (int ip = 0; ip < met->np; ip++) {
02940          met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
02941          met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
02942          met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
02943          met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
02944          met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
02945          met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
02946          met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
02947          met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
02948      }
02949  }
02950 }

```

5.19.2.28 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 2954 of file libtrac.c.

```

02955  {
02956
02957      double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
02958             dtdp, dudp, dvdp, latr, vort, pows[EP];
02959
02960      int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
02961
02962      /* Set powers... */
02963      for (ip = 0; ip < met->np; ip++)
02964          pows[ip] = pow(1000. / met->p[ip], 0.286);
02965
02966      /* Loop over grid points... */
02967      #pragma omp parallel for default(shared)
02968      private (ix, ix0, ix1, iy, iy0, iy1, latr, dx, dy, c0, c1, cr, vort, ip, ip0, ip1, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy, dtdp, dudp, dvdp)
02969      for (ix = 0; ix < met->nx; ix++) {
02970
02971          /* Set indices... */
02972          ix0 = GSL_MAX(ix - 1, 0);
02973          ix1 = GSL_MIN(ix + 1, met->nx - 1);
02974
02975          /* Loop over grid points... */
02976          for (iy = 0; iy < met->ny; iy++) {
02977
02978              /* Set indices... */
02979              iy0 = GSL_MAX(iy - 1, 0);
02980              iy1 = GSL_MIN(iy + 1, met->ny - 1);
02981
02982              /* Set auxiliary variables... */
02983              latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
02984              dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
02985              dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
02986              c0 = cos(met->lat[iy0] / 180. * M_PI);
02987              c1 = cos(met->lat[iy1] / 180. * M_PI);
02988              cr = cos(latr / 180. * M_PI);
02989              vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
02990
02991              /* Loop over grid points... */
02992              for (ip = 0; ip < met->np; ip++) {
02993
02994                  /* Get gradients in longitude... */
02995                  dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
02996                  dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
02997
02998                  /* Get gradients in latitude... */
02999                  dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03000                  dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03001
03002                  /* Set indices... */
03003                  ip0 = GSL_MAX(ip - 1, 0);
03004                  ip1 = GSL_MIN(ip + 1, met->np - 1);

```

```

03005      /* Get gradients in pressure... */
03006      dp0 = 100. * (met->p[ip] - met->p[ip0]);
03007      dp1 = 100. * (met->p[ip1] - met->p[ip]);
03008      if (ip != ip0 && ip != ip1) {
03009          denom = dp0 * dp1 * (dp0 + dp1);
03010          dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03011                - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03012                + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03013                / denom;
03014          dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03015                - dp1 * dp1 * met->u[ix][iy][ip0]
03016                + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03017                / denom;
03018          dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03019                - dp1 * dp1 * met->v[ix][iy][ip0]
03020                + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03021                / denom;
03022      } else {
03023          denom = dp0 + dp1;
03024          dtdp =
03025              (met->t[ix][iy][ip1] * pows[ip1] -
03026               met->t[ix][iy][ip0] * pows[ip0]) / denom;
03027          dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03028          dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03029      }
03030
03031      /* Calculate PV... */
03032      met->pv[ix][iy][ip] = (float)
03033          (1e6 * G0 *
03034           (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03035  }
03036  }
03037  }
03038
03039  /* Fix for polar regions... */
03040  #pragma omp parallel for default(shared) private(ix,ip)
03041  for (ix = 0; ix < met->nx; ix++)
03042      for (ip = 0; ip < met->np; ip++) {
03043          met->pv[ix][0][ip]
03044              = met->pv[ix][1][ip]
03045              = met->pv[ix][2][ip];
03046          met->pv[ix][met->ny - 1][ip]
03047              = met->pv[ix][met->ny - 2][ip]
03048              = met->pv[ix][met->ny - 3][ip];
03049      }
03050  }

```

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

Downsampling of meteorological data.

Definition at line 3054 of file libtrac.c.

```

03056      {
03057
03058      met_t *help;
03059
03060      float w, wsum;
03061
03062      int ip, ip2, ix, ix2, ix3, iy, iy2;
03063
03064      /* Check parameters... */
03065      if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03066          && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03067          return;
03068
03069      /* Allocate... */
03070      ALLOC(help, met_t, 1);
03071
03072      /* Copy data... */
03073      help->nx = met->nx;
03074      help->ny = met->ny;
03075      help->np = met->np;
03076      memcpy(help->lon, met->lon, sizeof(met->lon));
03077      memcpy(help->lat, met->lat, sizeof(met->lat));
03078      memcpy(help->p, met->p, sizeof(met->p));
03079
03080      /* Smoothing... */
03081      for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03082          for (iy = 0; iy < met->ny; iy += ctl->met_dy) {

```

```

03083     for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03084         help->ps[ix][iy] = 0;
03085         help->zs[ix][iy] = 0;
03086         help->t[ix][iy][ip] = 0;
03087         help->u[ix][iy][ip] = 0;
03088         help->v[ix][iy][ip] = 0;
03089         help->w[ix][iy][ip] = 0;
03090         help->h2o[ix][iy][ip] = 0;
03091         help->o3[ix][iy][ip] = 0;
03092         help->lwc[ix][iy][ip] = 0;
03093         help->iwc[ix][iy][ip] = 0;
03094         wsum = 0;
03095         for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03096             ix3 = ix2;
03097             if (ix3 < 0)
03098                 ix3 += met->nx;
03099             else if (ix3 >= met->nx)
03100                 ix3 -= met->nx;
03101
03102             for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03103                  iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03104                 for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03105                      ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03106                     w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
03107                         * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
03108                         * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
03109                     help->ps[ix][iy] += w * met->ps[ix3][iy2];
03110                     help->zs[ix][iy] += w * met->zs[ix3][iy2];
03111                     help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03112                     help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03113                     help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03114                     help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03115                     help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03116                     help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03117                     help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03118                     help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03119                     wsum += w;
03120                 }
03121             }
03122         help->ps[ix][iy] /= wsum;
03123         help->zs[ix][iy] /= wsum;
03124         help->t[ix][iy][ip] /= wsum;
03125         help->u[ix][iy][ip] /= wsum;
03126         help->v[ix][iy][ip] /= wsum;
03127         help->w[ix][iy][ip] /= wsum;
03128         help->h2o[ix][iy][ip] /= wsum;
03129         help->o3[ix][iy][ip] /= wsum;
03130         help->lwc[ix][iy][ip] /= wsum;
03131         help->iwc[ix][iy][ip] /= wsum;
03132     }
03133 }
03134 }
03135
03136 /* Downsampling... */
03137 met->nx = 0;
03138 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03139     met->lon[met->nx] = help->lon[ix];
03140     met->ny = 0;
03141     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03142         met->lat[met->ny] = help->lat[iy];
03143         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03144         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03145         met->np = 0;
03146         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03147             met->p[met->np] = help->p[ip];
03148             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03149             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03150             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03151             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03152             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03153             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03154             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03155             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03156             met->np++;
03157         }
03158         met->ny++;
03159     }
03160     met->nx++;
03161 }
03162
03163 /* Free... */
03164 free(help);
03165 }

```

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

Read surface data.

Definition at line 3169 of file [libtrac.c](#).

```

03171         {
03172
03173     int ix, iy;
03174
03175     /* Read surface pressure... */
03176     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03177         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03178             ERRMSG("Cannot not read surface pressure data!");
03179             for (ix = 0; ix < met->nx; ix++)
03180                 for (iy = 0; iy < met->ny; iy++)
03181                     met->ps[ix][iy] = (float) met->p[0];
03182         } else {
03183             for (iy = 0; iy < met->ny; iy++)
03184                 for (ix = 0; ix < met->nx; ix++)
03185                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03186         }
03187     }
03188
03189     /* Read geopotential height at the surface... */
03190     if (!read_met_help_2d
03191         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03192         if (!read_met_help_2d
03193             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03194             ERRMSG("Cannot read surface geopotential height!");
03195 }

```

Here is the call graph for this function:

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

Calculate tropopause pressure.

Definition at line 3199 of file [libtrac.c](#).

```

03201         {
03202
03203     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03204           th2[200], z[EP], z2[200];
03205
03206     int found, ix, iy, iz, iz2;
03207
03208     /* Get altitude and pressure profiles... */
03209     for (iz = 0; iz < met->np; iz++)
03210         z[iz] = Z(met->p[iz]);
03211     for (iz = 0; iz <= 190; iz++) {
03212         z2[iz] = 4.5 + 0.1 * iz;
03213         p2[iz] = P(z2[iz]);
03214     }
03215
03216     /* Do not calculate tropopause... */
03217     if (ctl->met_tropo == 0)
03218         for (ix = 0; ix < met->nx; ix++)

```

```

03219     for (iy = 0; iy < met->ny; iy++)
03220         met->pt[ix][iy] = GSL_NAN;
03221
03222     /* Use tropopause climatology... */
03223     else if (ctl->met_tropo == 1) {
03224 #pragma omp parallel for default(shared) private(ix,iy)
03225         for (ix = 0; ix < met->nx; ix++)
03226             for (iy = 0; iy < met->ny; iy++)
03227                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03228     }
03229
03230     /* Use cold point... */
03231     else if (ctl->met_tropo == 2) {
03232
03233         /* Loop over grid points... */
03234 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03235         for (ix = 0; ix < met->nx; ix++)
03236             for (iy = 0; iy < met->ny; iy++) {
03237
03238                 /* Interpolate temperature profile... */
03239                 for (iz = 0; iz < met->np; iz++)
03240                     t[iz] = met->t[ix][iy][iz];
03241                 spline(z, t, met->np, z2, t2, 171);
03242
03243                 /* Find minimum... */
03244                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03245                 if (iz > 0 && iz < 170)
03246                     met->pt[ix][iy] = (float) p2[iz];
03247                 else
03248                     met->pt[ix][iy] = GSL_NAN;
03249             }
03250     }
03251
03252     /* Use WMO definition... */
03253     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03254
03255         /* Loop over grid points... */
03256 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03257         for (ix = 0; ix < met->nx; ix++)
03258             for (iy = 0; iy < met->ny; iy++) {
03259
03260                 /* Interpolate temperature profile... */
03261                 for (iz = 0; iz < met->np; iz++)
03262                     t[iz] = met->t[ix][iy][iz];
03263                 spline(z, t, met->np, z2, t2, 191);
03264
03265                 /* Find 1st tropopause... */
03266                 met->pt[ix][iy] = GSL_NAN;
03267                 for (iz = 0; iz <= 170; iz++) {
03268                     found = 1;
03269                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03270                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03271                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03272                             found = 0;
03273                             break;
03274                         }
03275                     if (found) {
03276                         if (iz > 0 && iz < 170)
03277                             met->pt[ix][iy] = (float) p2[iz];
03278                         break;
03279                     }
03280                 }
03281
03282                 /* Find 2nd tropopause... */
03283                 if (ctl->met_tropo == 4) {
03284                     met->pt[ix][iy] = GSL_NAN;
03285                     for (; iz <= 170; iz++) {
03286                         found = 1;
03287                         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03288                             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03289                                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
03290                                 found = 0;
03291                                 break;
03292                             }
03293                         if (found)
03294                             break;
03295                     }
03296                     for (; iz <= 170; iz++) {
03297                         found = 1;
03298                         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03299                             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03300                                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03301                                 found = 0;
03302                                 break;
03303                             }
03304                         if (found) {
03305                             if (iz > 0 && iz < 170)

```

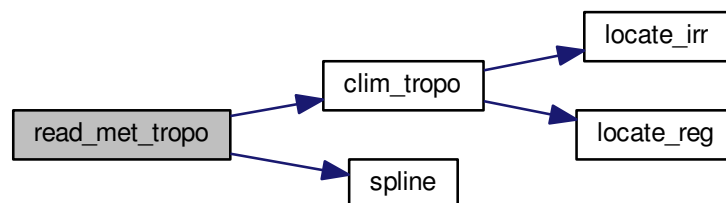


```

03306         met->pt[ix][iy] = (float) p2[iz];
03307         break;
03308     }
03309 }
03310 }
03311 }
03312 }
03313
03314 /* Use dynamical tropopause... */
03315 else if (ctl->met_tropo == 5) {
03316
03317     /* Loop over grid points... */
03318 #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03319     for (ix = 0; ix < met->nx; ix++)
03320         for (iy = 0; iy < met->ny; iy++) {
03321
03322             /* Interpolate potential vorticity profile... */
03323             for (iz = 0; iz < met->np; iz++)
03324                 pv[iz] = met->pv[ix][iy][iz];
03325             spline(z, pv, met->np, z2, pv2, 171);
03326
03327             /* Interpolate potential temperature profile... */
03328             for (iz = 0; iz < met->np; iz++)
03329                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03330             spline(z, th, met->np, z2, th2, 171);
03331
03332             /* Find dynamical tropopause 3.5 PVU + 380 K */
03333             met->pt[ix][iy] = GSL_NAN;
03334             for (iz = 0; iz <= 170; iz++)
03335                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03336                     if (iz > 0 && iz < 170)
03337                         met->pt[ix][iy] = (float) p2[iz];
03338                     break;
03339                 }
03340         }
03341     }
03342
03343     else
03344         ERRMSG("Cannot calculate tropopause!");
03345 }

```

Here is the call graph for this function:



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

Read a control parameter from file or command line.

Definition at line 3349 of file libtrac.c.

```

03356     {
03357
03358     FILE *in = NULL;
03359
03360     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],

```

```

03361     msg[2 * LEN], rvarname[LEN], rval[LEN];
03362
03363     int contain = 0, i;
03364
03365     /* Open file... */
03366     if (filename[strlen(filename) - 1] != '-')
03367         if (!(in = fopen(filename, "r")))
03368             ERRMSG("Cannot open file!");
03369
03370     /* Set full variable name... */
03371     if (arridx >= 0) {
03372         sprintf(fullname1, "%s[%d]", varname, arridx);
03373         sprintf(fullname2, "%s[*]", varname);
03374     } else {
03375         sprintf(fullname1, "%s", varname);
03376         sprintf(fullname2, "%s", varname);
03377     }
03378
03379     /* Read data... */
03380     if (in != NULL)
03381         while (fgets(line, LEN, in))
03382             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03383                 if (strcasecmp(rvarname, fullname1) == 0 ||
03384                     strcasecmp(rvarname, fullname2) == 0) {
03385                     contain = 1;
03386                     break;
03387                 }
03388     for (i = 1; i < argc - 1; i++)
03389         if (strcasecmp(argv[i], fullname1) == 0 ||
03390             strcasecmp(argv[i], fullname2) == 0) {
03391             sprintf(rval, "%s", argv[i + 1]);
03392             contain = 1;
03393             break;
03394         }
03395
03396     /* Close file... */
03397     if (in != NULL)
03398         fclose(in);
03399
03400     /* Check for missing variables... */
03401     if (!contain) {
03402         if (strlen(defvalue) > 0)
03403             sprintf(rval, "%s", defvalue);
03404         else {
03405             sprintf(msg, "Missing variable %s!\n", fullname1);
03406             ERRMSG(msg);
03407         }
03408     }
03409
03410     /* Write info... */
03411     printf("%s = %s\n", fullname1, rval);
03412
03413     /* Return values... */
03414     if (value != NULL)
03415         sprintf(value, "%s", rval);
03416     return atof(rval);
03417 }

```

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

Spline interpolation.

Definition at line 3421 of file [libtrac.c](#).

```

03427     {
03428
03429     gsl_interp_accel *acc;
03430
03431     gsl_spline *s;
03432
03433     /* Allocate... */
03434     acc = gsl_interp_accel_alloc();
03435     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03436
03437     /* Interpolate temperature profile... */
03438     gsl_spline_init(s, x, y, (size_t) n);
03439     for (int i = 0; i < n2; i++)
03440         if (x2[i] <= x[0])
03441             y2[i] = y[0];
03442         else if (x2[i] >= x[n - 1])

```

```
03443     y2[i] = y[n - 1];
03444     else
03445         y2[i] = gsl_spline_eval(s, x2[i], acc);
03446
03447     /* Free... */
03448     gsl_spline_free(s);
03449     gsl_interp_accel_free(acc);
03450 }
```

5.19.2.34 double stddev (double * data, int n)

Calculate standard deviation.

Definition at line 3454 of file [libtrac.c](#).

```
03456     {
03457
03458     if (n <= 0)
03459         return 0;
03460
03461     double avg = 0, rms = 0;
03462
03463     for (int i = 0; i < n; ++i)
03464         avg += data[i];
03465     avg /= n;
03466
03467     for (int i = 0; i < n; ++i)
03468         rms += SQR(data[i] - avg);
03469
03470     return sqrt(rms / (n - 1));
03471 }
```

5.19.2.35 void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double * jsec)

Convert date to seconds.

Definition at line 3475 of file [libtrac.c](#).

```
03483     {
03484
03485     struct tm t0, t1;
03486
03487     t0.tm_year = 100;
03488     t0.tm_mon = 0;
03489     t0.tm_mday = 1;
03490     t0.tm_hour = 0;
03491     t0.tm_min = 0;
03492     t0.tm_sec = 0;
03493
03494     t1.tm_year = year - 1900;
03495     t1.tm_mon = mon - 1;
03496     t1.tm_mday = day;
03497     t1.tm_hour = hour;
03498     t1.tm_min = min;
03499     t1.tm_sec = sec;
03500
03501     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03502 }
```

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

Measure wall-clock time.

Definition at line 3506 of file libtrac.c.

```

03509         {
03510
03511     static double starttime[NTIMER], runtime[NTIMER];
03512
03513     /* Check id... */
03514     if (id < 0 || id >= NTIMER)
03515         ERRMSG("Too many timers!");
03516
03517     /* Start timer... */
03518     if (mode == 1) {
03519         if (starttime[id] <= 0)
03520             starttime[id] = omp_get_wtime();
03521         else
03522             ERRMSG("Timer already started!");
03523     }
03524
03525     /* Stop timer... */
03526     else if (mode == 2) {
03527         if (starttime[id] > 0) {
03528             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
03529             starttime[id] = -1;
03530         }
03531     }
03532
03533     /* Print timer... */
03534     else if (mode == 3) {
03535         printf("%s = %.3f s\n", name, runtime[id]);
03536         runtime[id] = 0;
03537     }
03538 }

```

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

Write atmospheric data.

Definition at line 3542 of file libtrac.c.

```

03546         {
03547
03548     FILE *in, *out;
03549
03550     char line[LEN];
03551
03552     double r, t0, t1;
03553
03554     int ip, iq, year, mon, day, hour, min, sec;
03555
03556     /* Set time interval for output... */
03557     t0 = t - 0.5 * ctl->dt_mod;
03558     t1 = t + 0.5 * ctl->dt_mod;
03559
03560     /* Write info... */
03561     printf("Write atmospheric data: %s\n", filename);
03562
03563     /* Write ASCII data... */
03564     if (ctl->atm_type == 0) {
03565
03566         /* Check if gnuplot output is requested... */
03567         if (ctl->atm_gpfile[0] != '-') {
03568
03569             /* Create gnuplot pipe... */
03570             if (!(out = popen("gnuplot", "w")))
03571                 ERRMSG("Cannot create pipe to gnuplot!");
03572
03573             /* Set plot filename... */
03574             fprintf(out, "set out \"%s.png\"\n", filename);
03575
03576             /* Set time string... */
03577             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
03578             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",

```

```

03579         year, mon, day, hour, min);
03580
03581     /* Dump gnuplot file to pipe... */
03582     if (!(in = fopen(ctl->atm_gpfile, "r")))
03583         ERRMSG("Cannot open file!");
03584     while (fgets(line, LEN, in))
03585         fprintf(out, "%s", line);
03586     fclose(in);
03587 }
03588
03589 else {
03590
03591     /* Create file... */
03592     if (!(out = fopen(filename, "w")))
03593         ERRMSG("Cannot create file!");
03594 }
03595
03596 /* Write header... */
03597 fprintf(out,
03598         "# $1 = time [s]\n"
03599         "# $2 = altitude [km]\n"
03600         "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03601 for (iq = 0; iq < ctl->nq; iq++)
03602     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
03603             ctl->qnt_unit[iq]);
03604 fprintf(out, "\n");
03605
03606 /* Write data... */
03607 for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
03608
03609     /* Check time... */
03610     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
03611         continue;
03612
03613     /* Write output... */
03614     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
03615             atm->lon[ip], atm->lat[ip]);
03616     for (iq = 0; iq < ctl->nq; iq++) {
03617         fprintf(out, " ");
03618         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03619     }
03620     fprintf(out, "\n");
03621 }
03622
03623 /* Close file... */
03624 fclose(out);
03625 }
03626
03627 /* Write binary data... */
03628 else if (ctl->atm_type == 1) {
03629
03630     /* Create file... */
03631     if (!(out = fopen(filename, "w")))
03632         ERRMSG("Cannot create file!");
03633
03634     /* Write data... */
03635     FWRITE(&atm->np, int,
03636           1,
03637           out);
03638     FWRITE(atm->time, double,
03639           (size_t) atm->np,
03640           out);
03641     FWRITE(atm->p, double,
03642           (size_t) atm->np,
03643           out);
03644     FWRITE(atm->lon, double,
03645           (size_t) atm->np,
03646           out);
03647     FWRITE(atm->lat, double,
03648           (size_t) atm->np,
03649           out);
03650     for (iq = 0; iq < ctl->nq; iq++)
03651         FWRITE(atm->q[iq], double,
03652               (size_t) atm->np,
03653               out);
03654
03655     /* Close file... */
03656     fclose(out);
03657 }
03658
03659 /* Error... */
03660 else
03661     ERRMSG("Atmospheric data type not supported!");
03662 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 3666 of file libtrac.c.

```

03670     {
03671
03672     static FILE *in, *out;
03673
03674     static char line[LEN];
03675
03676     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
03677         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
03678
03679     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
03680
03681     /* Init... */
03682     if (t == ctl->t_start) {
03683
03684         /* Check quantity index for mass... */
03685         if (ctl->qnt_m < 0)
03686             ERRMSG("Need quantity mass!");
03687
03688         /* Open observation data file... */
03689         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
03690         if (!(in = fopen(ctl->csi_obsfile, "r")))
03691             ERRMSG("Cannot open file!");
03692
03693         /* Create new file... */
03694         printf("Write CSI data: %s\n", filename);
03695         if (!(out = fopen(filename, "w")))
03696             ERRMSG("Cannot create file!");
03697
03698         /* Write header... */
03699         fprintf(out,
03700             "# $1 = time [s]\n"
03701             "# $2 = number of hits (cx)\n"
03702             "# $3 = number of misses (cy)\n"
03703             "# $4 = number of false alarms (cz)\n"
03704             "# $5 = number of observations (cx + cy)\n"
03705             "# $6 = number of forecasts (cx + cz)\n"
03706             "# $7 = bias (forecasts/observations) [%%]\n"
03707             "# $8 = probability of detection (POD) [%%]\n"
03708             "# $9 = false alarm rate (FAR) [%%]\n"
03709             "# $10 = critical success index (CSI) [%%]\n\n");
03710     }
03711
03712     /* Set time interval... */
03713     t0 = t - 0.5 * ctl->dt_mod;
03714     t1 = t + 0.5 * ctl->dt_mod;
03715
03716     /* Initialize grid cells... */
03717     #pragma omp parallel for default(shared) private(ix,iy,iz)
03718     for (ix = 0; ix < ctl->csi_nx; ix++)
03719         for (iy = 0; iy < ctl->csi_ny; iy++)
03720             for (iz = 0; iz < ctl->csi_nz; iz++)
03721                 modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
03722
03723     /* Read observation data... */
03724     while (fgets(line, LEN, in)) {
03725
03726         /* Read data... */

```

```

03727     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln, &rln) !=
03728         5)
03729         continue;
03730
03731     /* Check time... */
03732     if (rt < t0)
03733         continue;
03734     if (rt > t1)
03735         break;
03736
03737     /* Calculate indices... */
03738     ix = (int) ((rln - ctl->csi_lon0)
03739                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03740     iy = (int) ((rln - ctl->csi_lat0)
03741                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03742     iz = (int) ((rz - ctl->csi_z0)
03743                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03744
03745     /* Check indices... */
03746     if (ix < 0 || ix >= ctl->csi_nx ||
03747         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03748         continue;
03749
03750     /* Get mean observation index... */
03751     obsmean[ix][iy][iz] += robs;
03752     obscount[ix][iy][iz]++;
03753 }
03754
03755 /* Analyze model data... */
03756 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03757 for (ip = 0; ip < atm->np; ip++) {
03758
03759     /* Check time... */
03760     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03761         continue;
03762
03763     /* Get indices... */
03764     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
03765                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03766     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
03767                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03768     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
03769                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03770
03771     /* Check indices... */
03772     if (ix < 0 || ix >= ctl->csi_nx ||
03773         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03774         continue;
03775
03776     /* Get total mass in grid cell... */
03777     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03778 }
03779
03780 /* Analyze all grid cells... */
03781 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
03782 for (ix = 0; ix < ctl->csi_nx; ix++)
03783     for (iy = 0; iy < ctl->csi_ny; iy++)
03784         for (iz = 0; iz < ctl->csi_nz; iz++) {
03785
03786             /* Calculate mean observation index... */
03787             if (obscount[ix][iy][iz] > 0)
03788                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
03789
03790             /* Calculate column density... */
03791             if (modmean[ix][iy][iz] > 0) {
03792                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
03793                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
03794                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
03795                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
03796                     * cos(lat * M_PI / 180.);
03797                 modmean[ix][iy][iz] /= (1e6 * area);
03798             }
03799
03800             /* Calculate CSI... */
03801             if (obscount[ix][iy][iz] > 0) {
03802                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03803                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03804                     cx++;
03805                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03806                     modmean[ix][iy][iz] < ctl->csi_modmin)
03807                     cy++;
03808                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
03809                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03810                     cz++;
03811             }
03812         }
03813

```

```

03814  /* Write output... */
03815  if (fmod(t, ctl->csi_dt_out) == 0) {
03816
03817      /* Write... */
03818      fprintf(out, "%.2f %d %d %d %d %d %g %g %g\n",
03819              t, cx, cy, cz, cx + cy, cx + cz,
03820              (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
03821              (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
03822              (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
03823              (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
03824
03825      /* Set counters to zero... */
03826      cx = cy = cz = 0;
03827  }
03828
03829  /* Close file... */
03830  if (t == ctl->t_stop)
03831      fclose(out);
03832 }

```

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

Write ensemble data.

Definition at line 3836 of file libtrac.c.

```

03840      {
03841
03842      static FILE *out;
03843
03844      static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
03845              t0, t1, x[NENS][3], xm[3];
03846
03847      static int ip, iq;
03848
03849      static size_t i, n;
03850
03851      /* Init... */
03852      if (t == ctl->t_start) {
03853
03854          /* Check quantities... */
03855          if (ctl->qnt_ens < 0)
03856              ERRMSG("Missing ensemble IDs!");
03857
03858          /* Create new file... */
03859          printf("Write ensemble data: %s\n", filename);
03860          if (!(out = fopen(filename, "w")))
03861              ERRMSG("Cannot create file!");
03862
03863          /* Write header... */
03864          fprintf(out,
03865                  "# $1 = time [s]\n"
03866                  "# $2 = altitude [km]\n"
03867                  "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03868          for (iq = 0; iq < ctl->nq; iq++)
03869              fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
03870                      ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03871          for (iq = 0; iq < ctl->nq; iq++)
03872              fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
03873                      ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03874          fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
03875      }
03876
03877      /* Set time interval... */
03878      t0 = t - 0.5 * ctl->dt_mod;
03879      t1 = t + 0.5 * ctl->dt_mod;
03880
03881      /* Init... */
03882      ens = GSL_NAN;
03883      n = 0;
03884
03885      /* Loop over air parcels... */
03886      for (ip = 0; ip < atm->np; ip++) {
03887
03888          /* Check time... */
03889          if (atm->time[ip] < t0 || atm->time[ip] > t1)
03890              continue;
03891
03892          /* Check ensemble id... */
03893          if (atm->q[ctl->qnt_ens][ip] != ens) {

```

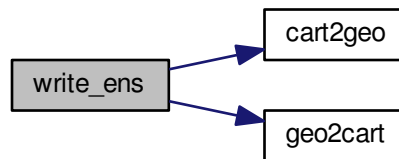


```

03894
03895     /* Write results... */
03896     if (n > 0) {
03897
03898         /* Get mean position... */
03899         xm[0] = xm[1] = xm[2] = 0;
03900         for (i = 0; i < n; i++) {
03901             xm[0] += x[i][0] / (double) n;
03902             xm[1] += x[i][1] / (double) n;
03903             xm[2] += x[i][2] / (double) n;
03904         }
03905         cart2geo(xm, &dummy, &lon, &lat);
03906         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
03907             lat);
03908
03909         /* Get quantity statistics... */
03910         for (iq = 0; iq < ctl->nq; iq++) {
03911             fprintf(out, " ");
03912             fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03913         }
03914         for (iq = 0; iq < ctl->nq; iq++) {
03915             fprintf(out, " ");
03916             fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03917         }
03918         fprintf(out, " %lu\n", n);
03919     }
03920
03921     /* Init new ensemble... */
03922     ens = atm->q[ctl->qnt_ens][ip];
03923     n = 0;
03924 }
03925
03926 /* Save data... */
03927 p[n] = atm->p[ip];
03928 geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
03929 for (iq = 0; iq < ctl->nq; iq++)
03930     q[iq][n] = atm->q[iq][ip];
03931 if ((++n) >= NENS)
03932     ERRMSG("Too many data points!");
03933 }
03934
03935 /* Write results... */
03936 if (n > 0) {
03937
03938     /* Get mean position... */
03939     xm[0] = xm[1] = xm[2] = 0;
03940     for (i = 0; i < n; i++) {
03941         xm[0] += x[i][0] / (double) n;
03942         xm[1] += x[i][1] / (double) n;
03943         xm[2] += x[i][2] / (double) n;
03944     }
03945     cart2geo(xm, &dummy, &lon, &lat);
03946     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
03947
03948     /* Get quantity statistics... */
03949     for (iq = 0; iq < ctl->nq; iq++) {
03950         fprintf(out, " ");
03951         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03952     }
03953     for (iq = 0; iq < ctl->nq; iq++) {
03954         fprintf(out, " ");
03955         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03956     }
03957     fprintf(out, " %lu\n", n);
03958 }
03959
03960 /* Close file... */
03961 if (t == ctl->t_stop)
03962     fclose(out);
03963 }

```

Here is the call graph for this function:



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

Write gridded data.

Definition at line 3967 of file libtrac.c.

```

03973     {
03974
03975     FILE *in, *out;
03976
03977     char line[LEN];
03978
03979     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
03980     area, rho_air, press, temp, cd, vmr, t0, t1, r, cw[3];
03981
03982     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
03983     ci[3];
03984
03985     /* Check dimensions... */
03986     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
03987         ERRMSG("Grid dimensions too large!");
03988
03989     /* Set time interval for output... */
03990     t0 = t - 0.5 * ctl->dt_mod;
03991     t1 = t + 0.5 * ctl->dt_mod;
03992
03993     /* Set grid box size... */
03994     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
03995     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
03996     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
03997
03998     /* Initialize grid... */
03999 #pragma omp parallel for default(shared) private(ix,iy,iz)
04000     for (ix = 0; ix < ctl->grid_nx; ix++)
04001     for (iy = 0; iy < ctl->grid_ny; iy++)
04002     for (iz = 0; iz < ctl->grid_nz; iz++) {
04003         mass[ix][iy][iz] = 0;
04004         np[ix][iy][iz] = 0;
04005     }
04006
04007     /* Average data... */
04008 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04009     for (ip = 0; ip < atm->np; ip++)
04010     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04011
04012         /* Get index... */
04013         ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04014         iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04015         iz = (int) ((atm->p[ip] - ctl->grid_z0) / dz);
04016
04017         /* Check indices... */
04018         if (ix < 0 || ix >= ctl->grid_nx ||
04019             iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04020             continue;
04021
04022         /* Add mass... */
04023         if (ctl->qnt_m >= 0)
04024             mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
  
```

```

04025     np[ix][iy][iz]++;
04026 }
04027
04028 /* Check if gnuplot output is requested... */
04029 if (ctl->grid_gpfile[0] != '-') {
04030
04031     /* Write info... */
04032     printf("Plot grid data: %s.png\n", filename);
04033
04034     /* Create gnuplot pipe... */
04035     if (!(out = popen("gnuplot", "w")))
04036         ERRMSG("Cannot create pipe to gnuplot!");
04037
04038     /* Set plot filename... */
04039     fprintf(out, "set out \"%s.png\"\n", filename);
04040
04041     /* Set time string... */
04042     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04043     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04044             year, mon, day, hour, min);
04045
04046     /* Dump gnuplot file to pipe... */
04047     if (!(in = fopen(ctl->grid_gpfile, "r")))
04048         ERRMSG("Cannot open file!");
04049     while (fgets(line, LEN, in))
04050         fprintf(out, "%s", line);
04051     fclose(in);
04052 }
04053
04054 else {
04055
04056     /* Write info... */
04057     printf("Write grid data: %s\n", filename);
04058
04059     /* Create file... */
04060     if (!(out = fopen(filename, "w")))
04061         ERRMSG("Cannot create file!");
04062 }
04063
04064 /* Write header... */
04065 fprintf(out,
04066         "# $1 = time [s]\n"
04067         "# $2 = altitude [km]\n"
04068         "# $3 = longitude [deg]\n"
04069         "# $4 = latitude [deg]\n"
04070         "# $5 = surface area [km^2]\n"
04071         "# $6 = layer width [km]\n"
04072         "# $7 = number of particles [l]\n"
04073         "# $8 = column density [kg/m^2]\n"
04074         "# $9 = volume mixing ratio [ppv]\n\n");
04075
04076 /* Write data... */
04077 for (ix = 0; ix < ctl->grid_nx; ix++) {
04078     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04079         fprintf(out, "\n");
04080     for (iy = 0; iy < ctl->grid_ny; iy++) {
04081         if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04082             fprintf(out, "\n");
04083         for (iz = 0; iz < ctl->grid_nz; iz++)
04084             if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04085
04086                 /* Set coordinates... */
04087                 z = ctl->grid_z0 + dz * (iz + 0.5);
04088                 lon = ctl->grid_lon0 + dlon * (ix + 0.5);
04089                 lat = ctl->grid_lat0 + dlat * (iy + 0.5);
04090
04091                 /* Get pressure and temperature... */
04092                 press = P(z);
04093                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04094                                   lat, &temp, ci, cw, 1);
04095
04096                 /* Calculate surface area... */
04097                 area = dlat * dlon * SQR(RE * M_PI / 180.)
04098                     * cos(lat * M_PI / 180.);
04099
04100                 /* Calculate column density... */
04101                 cd = mass[ix][iy][iz] / (1e6 * area);
04102
04103                 /* Calculate volume mixing ratio... */
04104                 rho_air = 100. * press / (RA * temp);
04105                 vmr = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
04106                     / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;
04107
04108                 /* Write output... */
04109                 fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
04110                         t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
04111             }

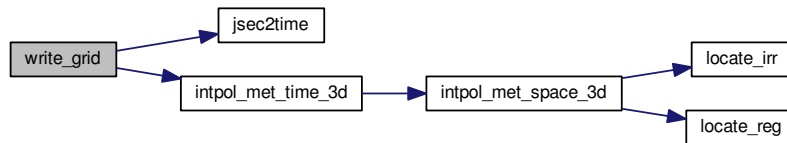
```

```

04112     }
04113 }
04114
04115 /* Close file... */
04116 fclose(out);
04117 }

```

Here is the call graph for this function:



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

Write profile data.

Definition at line 4121 of file libtrac.c.

```

04127     {
04128
04129     static FILE *in, *out;
04130
04131     static char line[LEN];
04132
04133     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
04134         t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
04135         o3, cw[3];
04136
04137     static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];
04138
04139     /* Init... */
04140     if (t == ctl->t_start) {
04141
04142         /* Check quantity index for mass... */
04143         if (ctl->qnt_m < 0)
04144             ERRMSG("Need quantity mass!");
04145
04146         /* Check dimensions... */
04147         if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04148             ERRMSG("Grid dimensions too large!");
04149
04150         /* Check molar mass... */
04151         if (ctl->molmass <= 0)
04152             ERRMSG("Specify molar mass!");
04153
04154         /* Open observation data file... */
04155         printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04156         if (!(in = fopen(ctl->prof_obsfile, "r")))
04157             ERRMSG("Cannot open file!");
04158
04159         /* Create new output file... */
04160         printf("Write profile data: %s\n", filename);
04161         if (!(out = fopen(filename, "w")))
04162             ERRMSG("Cannot create file!");
04163
04164         /* Write header... */
04165         fprintf(out,
04166             "# $1 = time [s]\n"
04167             "# $2 = altitude [km]\n"
04168             "# $3 = longitude [deg]\n"
04169             "# $4 = latitude [deg]\n"
04170             "# $5 = pressure [hPa]\n"
04171             "# $6 = temperature [K]\n"
04172             "# $7 = volume mixing ratio [ppv]\n"
04173             "# $8 = H2O volume mixing ratio [ppv]\n"

```

```

04174         "# $9 = O3 volume mixing ratio [ppv]\n"
04175         "# $10 = observed BT index [K]\n");
04176
04177     /* Set grid box size... */
04178     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04179     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04180     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04181 }
04182
04183 /* Set time interval... */
04184 t0 = t - 0.5 * ctl->dt_mod;
04185 t1 = t + 0.5 * ctl->dt_mod;
04186
04187 /* Initialize... */
04188 #pragma omp parallel for default(shared) private(ix,iy,iz)
04189 for (ix = 0; ix < ctl->prof_nx; ix++)
04190     for (iy = 0; iy < ctl->prof_ny; iy++) {
04191         obsmean[ix][iy] = 0;
04192         obscount[ix][iy] = 0;
04193         for (iz = 0; iz < ctl->prof_nz; iz++)
04194             mass[ix][iy][iz] = 0;
04195     }
04196
04197 /* Read observation data... */
04198 while (fgets(line, LEN, in)) {
04199
04200     /* Read data... */
04201     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04202         5)
04203         continue;
04204
04205     /* Check time... */
04206     if (rt < t0)
04207         continue;
04208     if (rt > t1)
04209         break;
04210
04211     /* Calculate indices... */
04212     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04213     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04214
04215     /* Check indices... */
04216     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04217         continue;
04218
04219     /* Get mean observation index... */
04220     obsmean[ix][iy] += robs;
04221     obscount[ix][iy]++;
04222 }
04223
04224 /* Analyze model data... */
04225 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04226 for (ip = 0; ip < atm->np; ip++) {
04227
04228     /* Check time... */
04229     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04230         continue;
04231
04232     /* Get indices... */
04233     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04234     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04235     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04236
04237     /* Check indices... */
04238     if (ix < 0 || ix >= ctl->prof_nx ||
04239         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04240         continue;
04241
04242     /* Get total mass in grid cell... */
04243     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04244 }
04245
04246 /* Extract profiles... */
04247 for (ix = 0; ix < ctl->prof_nx; ix++)
04248     for (iy = 0; iy < ctl->prof_ny; iy++)
04249         if (obscount[ix][iy] > 0) {
04250
04251             /* Check profile... */
04252             okay = 0;
04253             for (iz = 0; iz < ctl->prof_nz; iz++)
04254                 if (mass[ix][iy][iz] > 0) {
04255                     okay = 1;
04256                     break;
04257                 }
04258             if (!okay)
04259                 continue;
04260

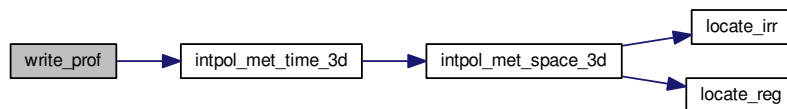
```

```

04261      /* Write output... */
04262      fprintf(out, "\n");
04263
04264      /* Loop over altitudes... */
04265      for (iz = 0; iz < ctl->prof_nz; iz++) {
04266
04267          /* Set coordinates... */
04268          z = ctl->prof_z0 + dz * (iz + 0.5);
04269          lon = ctl->prof_lon0 + dlon * (ix + 0.5);
04270          lat = ctl->prof_lat0 + dlat * (iy + 0.5);
04271
04272          /* Get pressure and temperature... */
04273          press = P(z);
04274          intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04275                          lat, &temp, ci, cw, 1);
04276          intpol_met_time_3d(met0, met0->h2o, met1, met1->
04277                          h2o, t, press, lon,
04278                          lat, &h2o, ci, cw, 0);
04279          intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
04280                          lat, &o3, ci, cw, 0);
04281
04282          /* Calculate surface area... */
04283          area = dlat * dlon * SQR(M_PI * RE / 180.)
04284              * cos(lat * M_PI / 180.);
04285
04286          /* Calculate volume mixing ratio... */
04287          rho_air = 100. * press / (RA * temp);
04288          vmr = MA / ctl->molmass * mass[ix][iy][iz]
04289              / (rho_air * area * dz * 1e9);
04290
04291          /* Write output... */
04292          fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
04293              t, z, lon, lat, press, temp, vmr, h2o, o3,
04294              obsmean[ix][iy] / obscount[ix][iy]);
04295      }
04296
04297      /* Close file... */
04298      if (t == ctl->t_stop)
04299          fclose(out);
04300 }

```

Here is the call graph for this function:



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

Write station data.

Definition at line 4304 of file libtrac.c.

```

04308      {
04309
04310          static FILE *out;
04311
04312          static double rmax2, t0, t1, x0[3], x1[3];
04313
04314          /* Init... */
04315          if (t == ctl->t_start) {
04316
04317              /* Write info... */
04318              printf("Write station data: %s\n", filename);
04319
04320              /* Create new file... */

```

```

04321     if (!(out = fopen(filename, "w")))
04322         ERRMSG("Cannot create file!");
04323
04324     /* Write header... */
04325     fprintf(out,
04326             "# $1 = time [s]\n"
04327             "# $2 = altitude [km]\n"
04328             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04329     for (int iq = 0; iq < ctl->nq; iq++)
04330         fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
04331                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04332     fprintf(out, "\n");
04333
04334     /* Set geolocation and search radius... */
04335     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
04336     rmax2 = SQR(ctl->stat_r);
04337 }
04338
04339 /* Set time interval for output... */
04340 t0 = t - 0.5 * ctl->dt_mod;
04341 t1 = t + 0.5 * ctl->dt_mod;
04342
04343 /* Loop over air parcels... */
04344 for (int ip = 0; ip < atm->np; ip++) {
04345
04346     /* Check time... */
04347     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04348         continue;
04349
04350     /* Check station flag... */
04351     if (ctl->qnt_stat >= 0)
04352         if (atm->q[ctl->qnt_stat][ip])
04353             continue;
04354
04355     /* Get Cartesian coordinates... */
04356     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04357
04358     /* Check horizontal distance... */
04359     if (DIST2(x0, x1) > rmax2)
04360         continue;
04361
04362     /* Set station flag... */
04363     if (ctl->qnt_stat >= 0)
04364         atm->q[ctl->qnt_stat][ip] = 1;
04365
04366     /* Write data... */
04367     fprintf(out, "%.2f %g %g %g",
04368             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
04369     for (int iq = 0; iq < ctl->nq; iq++) {
04370         fprintf(out, " ");
04371         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04372     }
04373     fprintf(out, "\n");
04374 }
04375
04376 /* Close file... */
04377 if (t == ctl->t_stop)
04378     fclose(out);
04379 }

```

Here is the call graph for this function:



5.20 libtrac.c

```
00001 /*
```

```

00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2020 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  /*****
00028
00029  void cart2geo(
00030      double *x,
00031      double *z,
00032      double *lon,
00033      double *lat) {
00034
00035      double radius = NORM(x);
00036      *lat = asin(x[2] / radius) * 180. / M_PI;
00037      *lon = atan2(x[1], x[0]) * 180. / M_PI;
00038      *z = radius - RE;
00039  }
00040
00041  /*****
00042
00043  // int isfinite(
00044  //     const double x) {
00045  //     const double y = x - x;
00046  //     int status = (y == y);
00047  //     return status;
00048  // }
00049
00050  /*****
00051
00052  static double clim_hno3_secs[12] = {
00053      1209600.00, 3888000.00, 6393600.00,
00054      9072000.00, 11664000.00, 14342400.00,
00055      16934400.00, 19612800.00, 22291200.00,
00056      24883200.00, 27561600.00, 30153600.00
00057  };
00058
00059  #ifdef _OPENACC
00060  #pragma acc declare copyin(clim_hno3_secs)
00061  #endif
00062
00063  static double clim_hno3_lats[18] = {
00064      -85, -75, -65, -55, -45, -35, -25, -15, -5,
00065      5, 15, 25, 35, 45, 55, 65, 75, 85
00066  };
00067
00068  #ifdef _OPENACC
00069  #pragma acc declare copyin(clim_hno3_lats)
00070  #endif
00071
00072  static double clim_hno3_ps[10] = {
00073      4.64159, 6.81292, 10, 14.678, 21.5443,
00074      31.6228, 46.4159, 68.1292, 100, 146.78
00075  };
00076
00077  #ifdef _OPENACC
00078  #pragma acc declare copyin(clim_hno3_ps)
00079  #endif
00080
00081  static double clim_hno3_var[12][18][10] = {
00082      {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00083       {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00084       {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00085       {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00086       {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00087       {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00088       {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00089       {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00090       {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00091       {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00092       {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00093       {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},

```



```

00094 {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00095 {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00096 {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00097 {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00098 {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00099 {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00100 {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00101 {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42}},
00102 {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00103 {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00104 {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00105 {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00106 {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00107 {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00108 {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00109 {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00110 {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00111 {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00112 {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00113 {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00114 {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00115 {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00116 {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00117 {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00118 {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00119 {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00120 {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00121 {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00122 {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00123 {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00124 {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00125 {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00126 {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00127 {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00128 {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00129 {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00130 {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00131 {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00132 {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00133 {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00134 {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00135 {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00136 {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00137 {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00138 {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00139 {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00140 {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00141 {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00142 {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
00143 {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
00144 {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
00145 {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
00146 {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
00147 {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
00148 {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
00149 {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
00150 {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
00151 {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
00152 {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
00153 {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
00154 {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
00155 {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
00156 {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
00157 {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
00158 {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
00159 {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
00160 {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
00161 {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
00162 {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
00163 {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
00164 {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
00165 {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
00166 {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
00167 {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
00168 {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
00169 {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
00170 {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
00171 {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6},
00172 {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
00173 {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
00174 {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
00175 {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
00176 {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
00177 {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
00178 {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
00179 {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
00180 {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},

```

00181 {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
 00182 {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
 00183 {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
 00184 {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
 00185 {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
 00186 {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
 00187 {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
 00188 {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
 00189 {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91},
 00190 {3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
 00191 {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
 00192 {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
 00193 {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
 00194 {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
 00195 {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
 00196 {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
 00197 {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
 00198 {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
 00199 {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
 00200 {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
 00201 {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
 00202 {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
 00203 {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
 00204 {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
 00205 {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
 00206 {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
 00207 {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62},
 00208 {5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
 00209 {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
 00210 {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
 00211 {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
 00212 {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
 00213 {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
 00214 {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
 00215 {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
 00216 {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
 00217 {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
 00218 {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
 00219 {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
 00220 {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
 00221 {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
 00222 {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
 00223 {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
 00224 {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
 00225 {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55},
 00226 {1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
 00227 {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
 00228 {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
 00229 {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
 00230 {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
 00231 {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
 00232 {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
 00233 {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
 00234 {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
 00235 {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
 00236 {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
 00237 {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
 00238 {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
 00239 {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
 00240 {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
 00241 {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
 00242 {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
 00243 {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65},
 00244 {0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
 00245 {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
 00246 {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
 00247 {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
 00248 {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
 00249 {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
 00250 {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
 00251 {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
 00252 {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
 00253 {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
 00254 {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
 00255 {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
 00256 {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
 00257 {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
 00258 {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
 00259 {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
 00260 {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
 00261 {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8},
 00262 {0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
 00263 {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
 00264 {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
 00265 {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
 00266 {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
 00267 {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},

```

00268     {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
00269     {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
00270     {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
00271     {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
00272     {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
00273     {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
00274     {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
00275     {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
00276     {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
00277     {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
00278     {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
00279     {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05},
00280     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
00281     {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
00282     {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
00283     {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
00284     {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
00285     {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
00286     {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
00287     {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
00288     {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
00289     {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
00290     {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
00291     {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
00292     {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
00293     {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
00294     {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
00295     {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
00296     {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
00297     {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
00298 };
00299
00300 #ifdef _OPENACC
00301 #pragma acc declare copyin(clim_hno3_var)
00302 #endif
00303
00304 double clim_hno3(
00305     double t,
00306     double lat,
00307     double p) {
00308
00309     /* Get seconds since begin of year... */
00310     double sec = FMOD(t, 365.25 * 86400.);
00311     while (sec < 0)
00312         sec += 365.25 * 86400.;
00313
00314     /* Check pressure... */
00315     if (p < clim_hno3_ps[0])
00316         p = clim_hno3_ps[0];
00317     else if (p > clim_hno3_ps[9])
00318         p = clim_hno3_ps[9];
00319
00320     /* Get indices... */
00321     int isec = locate_irr(clim_hno3_secs, 12, sec);
00322     int ilat = locate_reg(clim_hno3_lats, 18, lat);
00323     int ip = locate_irr(clim_hno3_ps, 10, p);
00324
00325     /* Interpolate HNO3 climatology (Froidevaux et al., 2015)... */
00326     double aux00 = LIN(clim_hno3_ps[ip],
00327         clim_hno3_var[isec][ilat][ip],
00328         clim_hno3_ps[ip + 1],
00329         clim_hno3_var[isec][ilat][ip + 1], p);
00330     double aux01 = LIN(clim_hno3_ps[ip],
00331         clim_hno3_var[isec][ilat + 1][ip],
00332         clim_hno3_ps[ip + 1],
00333         clim_hno3_var[isec][ilat + 1][ip + 1], p);
00334     double aux10 = LIN(clim_hno3_ps[ip],
00335         clim_hno3_var[isec + 1][ilat][ip],
00336         clim_hno3_ps[ip + 1],
00337         clim_hno3_var[isec + 1][ilat][ip + 1], p);
00338     double aux11 = LIN(clim_hno3_ps[ip],
00339         clim_hno3_var[isec + 1][ilat + 1][ip],
00340         clim_hno3_ps[ip + 1],
00341         clim_hno3_var[isec + 1][ilat + 1][ip + 1], p);
00342     aux00 = LIN(clim_hno3_lats[ilat], aux00,
00343         clim_hno3_lats[ilat + 1], aux01, lat);
00344     aux11 = LIN(clim_hno3_lats[ilat], aux10,
00345         clim_hno3_lats[ilat + 1], aux11, lat);
00346     return LIN(clim_hno3_secs[isec], aux00,
00347         clim_hno3_secs[isec + 1], aux11, sec);
00348 }
00349
00350 /*****
00351
00352 static double clim_oh_secs[12] = {
00353     1209600.00, 3888000.00, 6393600.00,
00354     9072000.00, 11664000.00, 14342400.00,

```

```
00355 16934400.00, 19612800.00, 22291200.00,
00356 24883200.00, 27561600.00, 30153600.00
00357 };
00358
00359 #ifndef _OPENACC
00360 #pragma acc declare copyin(clim_oh_secs)
00361 #endif
00362
00363 static double clim_oh_lats[18] = {
00364 -85, -75, -65, -55, -45, -35, -25, -15, -5,
00365 5, 15, 25, 35, 45, 55, 65, 75, 85
00366 };
00367
00368 #ifndef _OPENACC
00369 #pragma acc declare copyin(clim_oh_lats)
00370 #endif
00371
00372 static double clim_oh_ps[34] = {
00373 0.17501, 0.233347, 0.31113, 0.41484, 0.553119, 0.737493, 0.983323,
00374 1.3111, 1.74813, 2.33084, 3.10779, 4.14372, 5.52496, 7.36661, 9.82214,
00375 13.0962, 17.4616, 23.2821, 31.0428, 41.3904, 55.1872, 73.583, 98.1107,
00376 130.814, 174.419, 232.559, 310.078, 413.438, 551.25, 735, 789.809,
00377 848.705, 911.993, 980
00378 };
00379
00380 #ifndef _OPENACC
00381 #pragma acc declare copyin(clim_oh_ps)
00382 #endif
00383
00384 static double clim_oh_var[12][18][34] = {
00385 {{6.422, 6.418, 7.221, 8.409, 9.768, 11.22, 12.65, 13.68, 14.03,
00386 13.06, 11.01, 8.791, 7.096, 6.025, 5.135, 4.057, 2.791, 1.902,
00387 1.318, 0.9553, 0.7083, 0.5542, 0.5145, 0.5485, 0.6292, 0.5982, 1.716,
00388 1.111, 0.9802, 0.6707, 0.5235, 0.4476, 0.3783, 0.3091}},
00389 {6.311, 6.394, 7.2, 8.349, 9.664, 11.02, 12.21, 13.06, 13.28,
00390 12.42, 10.59, 8.552, 6.944, 5.862, 4.948, 3.826, 2.689, 1.873,
00391 1.302, 0.9316, 0.7053, 0.5634, 0.508, 0.5207, 0.6166, 0.6789, 1.682,
00392 1.218, 1.079, 0.7621, 0.6662, 0.5778, 0.4875, 0.3997}},
00393 {5.851, 5.827, 6.393, 7.294, 8.322, 9.415, 10.46, 11.24, 11.59,
00394 11.13, 9.754, 7.97, 6.417, 5.331, 4.468, 3.512, 2.581, 1.855,
00395 1.336, 0.9811, 0.756, 0.6328, 0.6011, 0.6202, 0.7603, 0.8883, 1.303,
00396 1.124, 1.118, 0.9428, 0.8655, 0.8156, 0.7602, 0.6805}},
00397 {5.276, 5.158, 5.66, 6.463, 7.419, 8.488, 9.563, 10.45, 10.94,
00398 10.65, 9.465, 7.762, 6.204, 5.074, 4.209, 3.324, 2.511, 1.865,
00399 1.386, 1.066, 0.8521, 0.723, 0.6997, 0.7492, 0.8705, 0.8088, 1.22,
00400 1.192, 1.298, 1.096, 1.037, 0.9589, 0.8856, 0.7726}},
00401 {5.06, 4.919, 5.379, 6.142, 7.095, 8.156, 9.18, 10.09, 10.62,
00402 10.33, 9.123, 7.479, 5.967, 4.858, 3.987, 3.097, 2.342, 1.743,
00403 1.323, 1.044, 0.8598, 0.7596, 0.7701, 0.7858, 0.8741, 1.256, 1.266,
00404 1.418, 1.594, 1.247, 1.169, 1.111, 1.054, 0.9141}},
00405 {4.921, 4.759, 5.188, 5.936, 6.847, 7.871, 8.903, 9.805, 10.31,
00406 10, 8.818, 7.223, 5.757, 4.66, 3.75, 2.831, 2.1, 1.579,
00407 1.243, 1.017, 0.8801, 0.8193, 0.9409, 1.131, 0.7313, 1.201, 1.383,
00408 1.643, 1.751, 1.494, 1.499, 1.647, 1.934, 2.147}},
00409 {4.665, 4.507, 4.947, 5.652, 6.549, 7.573, 8.609, 9.499, 9.985,
00410 9.664, 8.478, 6.944, 5.519, 4.407, 3.511, 2.595, 1.917, 1.46,
00411 1.172, 1.009, 0.9372, 0.9439, 1.047, 1.219, 0.5712, 1.032, 1.342,
00412 1.716, 1.846, 1.551, 1.55, 1.686, 2.006, 2.235}},
00413 {4.424, 4.288, 4.678, 5.38, 6.271, 7.291, 8.324, 9.231, 9.678,
00414 9.264, 8.037, 6.532, 5.141, 4.037, 3.148, 2.319, 1.715, 1.318,
00415 1.078, 0.9647, 0.9327, 0.9604, 1.023, 0.4157, 0.4762, 1.04, 1.589,
00416 2.093, 1.957, 1.557, 1.52, 1.565, 1.776, 1.904}},
00417 {4.154, 3.996, 4.347, 5.004, 5.854, 6.869, 7.929, 8.837, 9.23,
00418 8.708, 7.447, 6.024, 4.761, 3.742, 2.898, 2.096, 1.55, 1.191,
00419 0.9749, 0.8889, 0.8745, 0.9004, 0.9648, 0.36, 0.4423, 0.973, 1.571,
00420 2.086, 1.971, 1.569, 1.537, 1.567, 1.74, 1.811}},
00421 {3.862, 3.738, 4.093, 4.693, 5.499, 6.481, 7.489, 8.328, 8.637,
00422 8.07, 6.863, 5.56, 4.438, 3.522, 2.736, 1.971, 1.441, 1.098,
00423 0.8945, 0.8155, 0.7965, 0.8013, 0.8582, 1.119, 0.4076, 0.8805, 1.446,
00424 1.977, 1.96, 1.713, 1.793, 2.055, 2.521, 2.776}},
00425 {3.619, 3.567, 3.943, 4.54, 5.295, 6.168, 7.033, 7.691, 7.884,
00426 7.326, 6.207, 5.032, 4.055, 3.263, 2.552, 1.871, 1.365, 1.022,
00427 0.8208, 0.7184, 0.6701, 0.6551, 0.6965, 0.7928, 0.3639, 0.6365, 0.9295,
00428 1.381, 1.847, 1.658, 1.668, 1.87, 2.245, 2.409}},
00429 {3.354, 3.395, 3.811, 4.39, 5.07, 5.809, 6.514, 7, 7.054,
00430 6.472, 5.463, 4.466, 3.649, 2.997, 2.396, 1.785, 1.289, 0.9304,
00431 0.7095, 0.5806, 0.5049, 0.4639, 0.4899, 0.5149, 0.5445, 0.5185, 0.7495,
00432 0.8662, 1.25, 1.372, 1.384, 1.479, 1.76, 1.874}},
00433 {3.008, 3.102, 3.503, 4.049, 4.657, 5.287, 5.845, 6.14, 6.032,
00434 5.401, 4.494, 3.665, 3.043, 2.575, 2.103, 1.545, 1.074, 0.7429,
00435 0.5514, 0.4313, 0.3505, 0.2957, 0.2688, 0.2455, 0.232, 0.3565, 0.4017,
00436 0.5063, 0.6618, 0.7621, 0.7915, 0.8372, 0.923, 0.9218}},
00437 {2.548, 2.725, 3.135, 3.637, 4.165, 4.666, 5.013, 5.056, 4.72,
00438 4.033, 3.255, 2.64, 2.24, 1.942, 1.555, 1.085, 0.7271, 0.502,
00439 0.3748, 0.2897, 0.2303, 0.19, 0.1645, 0.1431, 0.1215, 0.09467, 0.1442,
00440 0.1847, 0.2368, 0.2463, 0.2387, 0.2459, 0.2706, 0.2751}},
00441 {1.946, 2.135, 2.46, 2.831, 3.203, 3.504, 3.584, 3.37, 2.921,
```

```

00442     2.357, 1.865, 1.551, 1.392, 1.165, 0.8443, 0.5497, 0.3686, 0.2632,
00443     0.1978, 0.1509, 0.1197, 0.0992, 0.08402, 0.07068, 0.05652, 0.03962,
00444     0.03904,
00445     0.04357, 0.05302, 0.04795, 0.04441, 0.04296, 0.04446, 0.04576},
00446     {1.157, 1.285, 1.432, 1.546, 1.602, 1.556, 1.378, 1.15, 0.9351,
00447     0.7636, 0.6384, 0.5267, 0.4008, 0.2821, 0.2, 0.1336, 0.09109, 0.06557,
00448     0.05219, 0.04197, 0.03443, 0.03119, 0.02893, 0.02577, 0.02119, 0.01102,
00449     0.008897,
00450     0.00467, 0.004651, 0.004809, 0.004539, 0.004181, 0.003737, 0.002833},
00451     {0.07716, 0.06347, 0.05343, 0.04653, 0.0393, 0.03205, 0.02438, 0.01692,
00452     0.0115,
00453     0.007576, 0.00488, 0.002961, 0.001599, 0.001033, 0.001067, 0.001091,
00454     0.0005156, 0.0003818,
00455     0.0005061, 0.0005322, 0.0008027, 0.0008598, 0.0009114, 0.001112, 0.002042,
00456     0.0002528, 0.0005562,
00457     7.905e-06, 1.461e-07, 1.448e-07, 9.962e-08, 4.304e-08, 9.129e-17,
00458     1.36e-16},
00459     {2.613e-05, 3.434e-05, 3.646e-05, 5.101e-05, 8.027e-05, 0.0001172,
00460     9.886e-05, 1.933e-05, 3.14e-05,
00461     7.708e-05, 0.000136, 0.0001447, 0.0001049, 4.451e-05, 9.37e-06, 2.235e-06,
00462     1.034e-06, 4.87e-06,
00463     1.615e-05, 2.018e-05, 6.578e-05, 0.000178, 0.0002489, 0.0004818, 0.001231,
00464     0.0001402, 0.0004263,
00465     4.581e-07, 1.045e-12, 1.295e-13, 9.008e-14, 8.464e-14, 1.183e-13,
00466     2.471e-13}},
00467     {{5.459, 5.793, 6.743, 7.964, 9.289, 10.5, 11.39, 11.68, 11.14,
00468     9.663, 7.886, 6.505, 5.549, 4.931, 4.174, 3.014, 1.99, 1.339,
00469     0.9012, 0.6096, 0.4231, 0.3152, 0.2701, 0.2561, 0.2696, 0.2523, 0.7171,
00470     0.5333, 0.4876, 0.3218, 0.2536, 0.2178, 0.1861, 0.1546},
00471     {5.229, 5.456, 6.192, 7.112, 8.094, 9.038, 9.776, 10.16, 9.992,
00472     9, 7.493, 6.162, 5.154, 4.461, 3.788, 2.935, 2.058, 1.407,
00473     0.9609, 0.6738, 0.4989, 0.3927, 0.3494, 0.3375, 0.3689, 0.3866, 0.6716,
00474     0.7088, 0.6307, 0.4388, 0.3831, 0.3318, 0.2801, 0.2317},
00475     {4.712, 4.75, 5.283, 6.062, 6.943, 7.874, 8.715, 9.344, 9.516,
00476     8.913, 7.63, 6.223, 5.1, 4.346, 3.709, 2.982, 2.235, 1.605,
00477     1.142, 0.8411, 0.6565, 0.5427, 0.4942, 0.4907, 0.5447, 0.6331, 0.9356,
00478     0.7821, 0.7611, 0.663, 0.628, 0.5915, 0.5763, 0.5451},
00479     {4.621, 4.6, 5.091, 5.827, 6.688, 7.624, 8.529, 9.276, 9.631,
00480     9.219, 7.986, 6.499, 5.264, 4.401, 3.737, 2.996, 2.292, 1.69,
00481     1.237, 0.9325, 0.7325, 0.6093, 0.5742, 0.5871, 0.6446, 0.6139, 0.9845,
00482     0.9741, 1.044, 0.9091, 0.8661, 0.8218, 0.7617, 0.6884},
00483     {4.647, 4.573, 5.038, 5.766, 6.61, 7.534, 8.489, 9.252, 9.6,
00484     9.161, 7.958, 6.512, 5.259, 4.317, 3.547, 2.789, 2.13, 1.601,
00485     1.205, 0.9321, 0.7532, 0.6464, 0.6173, 0.5896, 0.5782, 1.014, 1.096,
00486     1.226, 1.387, 1.111, 1.042, 0.9908, 0.9408, 0.8311},
00487     {4.621, 4.534, 4.984, 5.693, 6.545, 7.49, 8.444, 9.177, 9.531,
00488     9.117, 7.928, 6.533, 5.27, 4.271, 3.431, 2.575, 1.902, 1.42,
00489     1.11, 0.9004, 0.7658, 0.6955, 0.7676, 0.9088, 0.8989, 1.028, 1.221,
00490     1.455, 1.583, 1.375, 1.376, 1.498, 1.744, 1.925},
00491     {4.514, 4.41, 4.837, 5.545, 6.416, 7.38, 8.287, 9.05, 9.416,
00492     9.022, 7.903, 6.496, 5.175, 4.111, 3.232, 2.38, 1.76, 1.335,
00493     1.068, 0.9227, 0.8515, 0.8511, 0.9534, 1.091, 0.4909, 0.9377, 1.241,
00494     1.592, 1.739, 1.478, 1.473, 1.597, 1.893, 2.117},
00495     {4.407, 4.264, 4.61, 5.263, 6.095, 7.046, 8.005, 8.805, 9.201,
00496     8.823, 7.705, 6.299, 4.964, 3.891, 3.046, 2.214, 1.641, 1.261,
00497     1.027, 0.922, 0.8759, 0.8893, 0.9782, 0.3707, 0.4349, 0.976, 1.523,
00498     2.021, 1.906, 1.524, 1.486, 1.53, 1.741, 1.869},
00499     {4.156, 4.007, 4.37, 4.987, 5.777, 6.719, 7.728, 8.578, 8.97,
00500     8.552, 7.409, 6.027, 4.731, 3.684, 2.814, 2.029, 1.501, 1.159,
00501     0.9542, 0.8666, 0.8191, 0.8371, 0.9704, 0.3324, 0.5634, 0.9279, 1.512,
00502     2.042, 1.951, 1.566, 1.535, 1.567, 1.739, 1.822},
00503     {3.98, 3.883, 4.232, 4.841, 5.594, 6.502, 7.497, 8.296, 8.631,
00504     8.161, 7.043, 5.703, 4.51, 3.548, 2.717, 1.951, 1.435, 1.107,
00505     0.9188, 0.8286, 0.772, 0.7564, 0.8263, 0.3026, 0.3854, 0.8743, 1.452,
00506     2.024, 2.012, 1.764, 1.85, 2.125, 2.621, 2.9},
00507     {3.877, 3.811, 4.167, 4.733, 5.462, 6.324, 7.144, 7.862, 8.14,
00508     7.695, 6.613, 5.393, 4.299, 3.42, 2.653, 1.947, 1.44, 1.1,
00509     0.8953, 0.7907, 0.7218, 0.6881, 0.7281, 0.8846, 0.3802, 0.6853, 1.025,
00510     1.593, 2.088, 1.846, 1.866, 2.108, 2.548, 2.737},
00511     {3.636, 3.634, 4.034, 4.637, 5.354, 6.148, 6.899, 7.495, 7.657,
00512     7.138, 6.107, 4.974, 3.983, 3.21, 2.558, 1.932, 1.439, 1.062,
00513     0.8104, 0.6632, 0.5721, 0.517, 0.5182, 0.501, 0.3937, 0.5565, 0.7675,
00514     0.9923, 1.455, 1.554, 1.567, 1.701, 2.077, 2.257},
00515     {3.366, 3.446, 3.885, 4.473, 5.16, 5.9, 6.557, 6.973, 6.973,
00516     6.379, 5.387, 4.385, 3.561, 2.941, 2.428, 1.848, 1.349, 0.9512,
00517     0.6956, 0.5432, 0.4437, 0.3733, 0.3297, 0.2972, 0.2518, 0.4613, 0.5078,
00518     0.6579, 0.8696, 0.9827, 1.028, 1.114, 1.275, 1.315},
00519     {2.975, 3.121, 3.567, 4.146, 4.797, 5.466, 6.003, 6.279, 6.087,
00520     5.372, 4.442, 3.614, 2.986, 2.521, 2.082, 1.532, 1.066, 0.7304,
00521     0.5316, 0.4123, 0.3314, 0.274, 0.2365, 0.2061, 0.1755, 0.1348, 0.2217,
00522     0.2854, 0.387, 0.4315, 0.4301, 0.4585, 0.5195, 0.5349},
00523     {2.434, 2.632, 3.054, 3.577, 4.156, 4.725, 5.1, 5.109, 4.7,
00524     3.958, 3.191, 2.588, 2.182, 1.866, 1.491, 1.034, 0.6936, 0.4774,
00525     0.3551, 0.2749, 0.2203, 0.1829, 0.1544, 0.1325, 0.1103, 0.07716, 0.08626,
00526     0.1037, 0.1455, 0.1418, 0.1351, 0.1339, 0.1407, 0.143},
00527     {1.798, 2.004, 2.333, 2.746, 3.164, 3.49, 3.572, 3.315, 2.833,
00528     2.269, 1.797, 1.485, 1.222, 0.9595, 0.6972, 0.4717, 0.3313, 0.2315,

```

00529 0.1727, 0.1407, 0.1204, 0.1116, 0.08791, 0.07567, 0.07432, 0.04138,
 00530 0.02942,
 00531 0.02407, 0.02969, 0.02808, 0.0261, 0.02364, 0.02079, 0.01623},
 00532 {1.01, 1.1, 1.194, 1.284, 1.32, 1.278, 1.136, 0.9619, 0.815,
 00533 0.7226, 0.6136, 0.472, 0.3124, 0.2056, 0.1448, 0.1041, 0.07233, 0.05587,
 00534 0.05634, 0.04788, 0.03971, 0.02542, 0.01735, 0.01522, 0.01475, 0.01076,
 00535 0.005235,
 00536 0.003546, 0.003113, 0.003146, 0.002902, 0.002635, 0.001966, 0.001156},
 00537 {0.04181, 0.03949, 0.03577, 0.03191, 0.02839, 0.02391, 0.01796, 0.01251,
 00538 0.008422,
 00539 0.005359, 0.003257, 0.001769, 0.001092, 0.0008405, 0.001744, 0.0003061,
 00540 0.0002956, 0.000334,
 00541 0.0004223, 0.0004062, 0.0003886, 0.0003418, 0.0003588, 0.0005709,
 00542 0.001368, 0.0001472, 0.0003282,
 00543 2.039e-06, 9.891e-13, 1.597e-13, 9.622e-14, 1.167e-13, 1.706e-13,
 00544 3.59e-13}},
 00545 {{3.361, 3.811, 4.35, 4.813, 5.132, 5.252, 5.019, 4.545, 3.913,
 00546 3.25, 2.739, 2.417, 2.071, 1.582, 1.059, 0.6248, 0.3911, 0.2807,
 00547 0.2095, 0.1539, 0.118, 0.0953, 0.08084, 0.07456, 0.07567, 0.07108, 0.1196,
 00548 0.08475, 0.0839, 0.05724, 0.04673, 0.0412, 0.03626, 0.03134},
 00549 {3.193, 3.459, 3.973, 4.578, 5.172, 5.677, 5.945, 5.913, 5.487,
 00550 4.686, 3.838, 3.217, 2.773, 2.409, 1.9, 1.33, 0.899, 0.64,
 00551 0.4734, 0.355, 0.277, 0.2245, 0.1928, 0.1794, 0.1806, 0.1934, 0.3268,
 00552 0.2217, 0.1969, 0.1375, 0.1206, 0.103, 0.08729, 0.07332},
 00553 {3.456, 3.617, 4.127, 4.778, 5.448, 6.083, 6.59, 6.858, 6.716,
 00554 6.004, 4.975, 4.092, 3.442, 2.986, 2.474, 1.87, 1.328, 0.9231,
 00555 0.6642, 0.5044, 0.4019, 0.336, 0.3049, 0.2996, 0.3227, 0.3571, 0.4295,
 00556 0.3789, 0.3663, 0.3286, 0.3128, 0.3015, 0.302, 0.2968},
 00557 {3.743, 3.817, 4.311, 4.948, 5.666, 6.417, 7.077, 7.528, 7.551,
 00558 6.893, 5.766, 4.721, 3.913, 3.326, 2.788, 2.144, 1.555, 1.093,
 00559 0.7905, 0.6101, 0.4943, 0.4214, 0.3965, 0.3958, 0.4205, 0.4093, 0.5808,
 00560 0.5876, 0.6359, 0.5956, 0.5731, 0.5485, 0.512, 0.4733},
 00561 {4.012, 4.012, 4.455, 5.113, 5.866, 6.661, 7.41, 7.98, 8.111,
 00562 7.509, 6.364, 5.223, 4.308, 3.602, 2.951, 2.23, 1.625, 1.168,
 00563 0.8748, 0.6916, 0.5758, 0.5122, 0.5053, 0.444, 0.4277, 0.7575, 0.7756,
 00564 0.8848, 1.034, 0.8886, 0.8352, 0.7965, 0.7562, 0.68},
 00565 {4.136, 4.113, 4.549, 5.187, 5.93, 6.754, 7.592, 8.298, 8.546,
 00566 8.022, 6.882, 5.668, 4.625, 3.778, 3.016, 2.236, 1.625, 1.193,
 00567 0.9253, 0.756, 0.6523, 0.598, 0.6186, 0.6287, 0.4804, 0.8007, 0.9732,
 00568 1.175, 1.329, 1.194, 1.193, 1.288, 1.47, 1.603},
 00569 {4.185, 4.119, 4.528, 5.186, 5.95, 6.801, 7.678, 8.404, 8.709,
 00570 8.272, 7.179, 5.922, 4.767, 3.814, 2.996, 2.187, 1.599, 1.203,
 00571 0.9596, 0.8307, 0.7617, 0.7489, 0.822, 0.9684, 0.4374, 0.7664, 1.044,
 00572 1.36, 1.525, 1.331, 1.334, 1.447, 1.685, 1.862},
 00573 {4.213, 4.111, 4.507, 5.117, 5.882, 6.773, 7.67, 8.464, 8.829,
 00574 8.438, 7.378, 6.088, 4.84, 3.802, 2.915, 2.114, 1.561, 1.195,
 00575 0.9731, 0.8773, 0.8296, 0.8295, 0.9193, 1.114, 0.3986, 0.8793, 1.394,
 00576 1.884, 1.786, 1.437, 1.403, 1.446, 1.64, 1.752},
 00577 {4.216, 4.092, 4.458, 5.04, 5.78, 6.67, 7.617, 8.47, 8.883,
 00578 8.531, 7.504, 6.163, 4.862, 3.792, 2.891, 2.063, 1.519, 1.174,
 00579 0.979, 0.8935, 0.8377, 0.8354, 0.9357, 0.3235, 0.4057, 0.9346, 1.53,
 00580 2.078, 1.951, 1.555, 1.525, 1.557, 1.737, 1.825},
 00581 {4.168, 4.054, 4.417, 5.011, 5.76, 6.655, 7.601, 8.453, 8.882,
 00582 8.539, 7.487, 6.143, 4.853, 3.809, 2.92, 2.088, 1.531, 1.181,
 00583 0.9861, 0.8955, 0.8297, 0.8116, 0.8945, 1.137, 0.4003, 0.9318, 1.557,
 00584 2.163, 2.095, 1.82, 1.931, 2.246, 2.823, 3.14},
 00585 {4.15, 4.078, 4.457, 5.044, 5.791, 6.692, 7.602, 8.343, 8.697,
 00586 8.324, 7.282, 5.969, 4.743, 3.764, 2.926, 2.138, 1.572, 1.197,
 00587 0.9808, 0.871, 0.7977, 0.764, 0.7663, 0.8665, 0.3969, 0.7472, 1.125,
 00588 1.78, 2.277, 1.98, 2.036, 2.365, 2.936, 3.186},
 00589 {4.028, 4.006, 4.421, 5.039, 5.793, 6.657, 7.515, 8.216, 8.5,
 00590 8.071, 7.01, 5.768, 4.638, 3.752, 3.021, 2.305, 1.714, 1.261,
 00591 0.9646, 0.7989, 0.6935, 0.6314, 0.6426, 0.5914, 0.4424, 0.6827, 0.917,
 00592 1.27, 1.864, 1.895, 1.922, 2.172, 2.772, 3.141},
 00593 {3.825, 3.864, 4.324, 4.97, 5.727, 6.565, 7.363, 7.978, 8.162,
 00594 7.628, 6.542, 5.369, 4.372, 3.601, 2.977, 2.292, 1.697, 1.211,
 00595 0.8808, 0.6852, 0.562, 0.4717, 0.4094, 0.366, 0.3093, 0.5639, 0.649,
 00596 0.8282, 1.144, 1.27, 1.346, 1.522, 1.831, 1.939},
 00597 {3.493, 3.611, 4.102, 4.736, 5.491, 6.333, 7.098, 7.615, 7.666,
 00598 7.006, 5.88, 4.791, 3.918, 3.283, 2.733, 2.099, 1.522, 1.066,
 00599 0.7595, 0.5845, 0.4736, 0.3918, 0.337, 0.2962, 0.2564, 0.2024, 0.3657,
 00600 0.4744, 0.6366, 0.76, 0.8059, 0.9015, 1.065, 1.126},
 00601 {3.167, 3.353, 3.857, 4.499, 5.225, 6.002, 6.699, 7.106, 6.98,
 00602 6.192, 5.094, 4.125, 3.4, 2.87, 2.382, 1.774, 1.246, 0.8549,
 00603 0.6089, 0.4666, 0.3762, 0.3082, 0.2643, 0.2283, 0.1942, 0.142, 0.1945,
 00604 0.2512, 0.3593, 0.4048, 0.4081, 0.4206, 0.4616, 0.4738},
 00605 {2.839, 3.07, 3.565, 4.17, 4.846, 5.529, 6.041, 6.194, 5.833,
 00606 5.009, 4.027, 3.239, 2.656, 2.227, 1.794, 1.282, 0.9003, 0.6219,
 00607 0.4405, 0.3353, 0.2672, 0.2205, 0.1979, 0.1911, 0.1846, 0.1044, 0.09845,
 00608 0.1069, 0.1547, 0.1611, 0.156, 0.1466, 0.1369, 0.1212},
 00609 {2.471, 2.75, 3.188, 3.712, 4.27, 4.77, 4.987, 4.802, 4.251,
 00610 3.473, 2.731, 2.217, 1.83, 1.498, 1.137, 0.7594, 0.5383, 0.3859,
 00611 0.2842, 0.2242, 0.1817, 0.1654, 0.1508, 0.1422, 0.1274, 0.07194, 0.05548,
 00612 0.04058, 0.05226, 0.05347, 0.04942, 0.04415, 0.03177, 0.01923},
 00613 {2.192, 2.463, 2.735, 2.958, 3.106, 3.134, 2.927, 2.561, 2.149,
 00614 1.825, 1.627, 1.429, 1.102, 0.766, 0.4943, 0.3166, 0.2184, 0.1471,
 00615 0.1087, 0.08747, 0.07338, 0.06265, 0.05554, 0.05001, 0.04253, 0.03353,

```

00616     0.01781,
00617     0.01332, 0.01371, 0.01531, 0.01438, 0.01296, 0.00982, 0.006101}},
00618     {{0.2905, 0.282, 0.2639, 0.2366, 0.2078, 0.1879, 0.1644, 0.1407, 0.1163,
00619     0.08898, 0.06233, 0.04005, 0.02438, 0.01319, 0.007679, 0.005681, 0.004482,
00620     0.004329,
00621     0.004242, 0.004414, 0.004655, 0.004515, 0.003998, 0.00342, 0.003203,
00622     0.001672, 0.00135,
00623     0.001376, 0.0004656, 0.0003538, 0.000242, 0.0001647, 0.0001078,
00624     5.702e-05}},
00625     {1.5, 1.685, 1.901, 2.105, 2.216, 2.203, 2.048, 1.855, 1.609,
00626     1.357, 1.162, 1.012, 0.8042, 0.573, 0.372, 0.223, 0.144, 0.1095,
00627     0.08876, 0.07152, 0.05912, 0.05031, 0.04379, 0.03983, 0.03757, 0.03481,
00628     0.04463,
00629     0.02747, 0.02318, 0.01802, 0.0155, 0.01347, 0.01136, 0.009558},
00630     {2.228, 2.464, 2.889, 3.347, 3.773, 4.098, 4.191, 4.074, 3.7,
00631     3.132, 2.591, 2.217, 1.929, 1.594, 1.177, 0.7799, 0.5288, 0.3786,
00632     0.2805, 0.2144, 0.1695, 0.1403, 0.1236, 0.1157, 0.1134, 0.1148, 0.1127,
00633     0.1059, 0.0982, 0.08934, 0.08645, 0.08666, 0.09037, 0.09339},
00634     {2.761, 2.942, 3.406, 3.981, 4.592, 5.177, 5.591, 5.699, 5.391,
00635     4.682, 3.883, 3.267, 2.804, 2.378, 1.889, 1.347, 0.9255, 0.6382,
00636     0.4661, 0.361, 0.2904, 0.2459, 0.2237, 0.2143, 0.2142, 0.1959, 0.2522,
00637     0.2552, 0.2714, 0.274, 0.2676, 0.2624, 0.2589, 0.2436},
00638     {3.207, 3.317, 3.769, 4.373, 5.05, 5.759, 6.356, 6.677, 6.533,
00639     5.82, 4.861, 4.056, 3.414, 2.865, 2.322, 1.708, 1.204, 0.8386,
00640     0.6195, 0.4861, 0.3997, 0.3459, 0.32, 0.2881, 0.251, 0.4026, 0.4166,
00641     0.4798, 0.5723, 0.5711, 0.5498, 0.5348, 0.5181, 0.4851},
00642     {3.509, 3.549, 3.992, 4.607, 5.301, 6.054, 6.784, 7.302, 7.372,
00643     6.753, 5.706, 4.725, 3.909, 3.223, 2.58, 1.894, 1.356, 0.9716,
00644     0.7397, 0.5944, 0.4998, 0.4401, 0.4193, 0.3649, 0.3204, 0.5317, 0.5992,
00645     0.7541, 0.9257, 0.9193, 0.9167, 0.976, 1.081, 1.167},
00646     {3.748, 3.717, 4.128, 4.736, 5.472, 6.31, 7.086, 7.725, 7.909,
00647     7.387, 6.323, 5.206, 4.217, 3.376, 2.638, 1.912, 1.382, 1.026,
00648     0.8072, 0.6849, 0.6123, 0.5755, 0.6121, 0.6279, 0.3644, 0.5863, 0.823,
00649     1.068, 1.241, 1.134, 1.134, 1.222, 1.397, 1.55},
00650     {3.966, 3.863, 4.239, 4.826, 5.583, 6.465, 7.317, 8.033, 8.34,
00651     7.916, 6.843, 5.63, 4.49, 3.524, 2.698, 1.933, 1.409, 1.074,
00652     0.8747, 0.7756, 0.7211, 0.7103, 0.7513, 0.3147, 0.3627, 0.7582, 1.206,
00653     1.64, 1.639, 1.321, 1.285, 1.334, 1.496, 1.611},
00654     {4.184, 4.066, 4.42, 4.981, 5.712, 6.597, 7.502, 8.305, 8.719,
00655     8.382, 7.314, 6.02, 4.76, 3.702, 2.787, 1.993, 1.461, 1.13,
00656     0.9379, 0.8456, 0.7865, 0.7693, 0.835, 0.3132, 0.4006, 0.9192, 1.486,
00657     1.956, 1.836, 1.485, 1.462, 1.502, 1.65, 1.74},
00658     {4.304, 4.184, 4.566, 5.178, 5.931, 6.823, 7.761, 8.572, 9.013,
00659     8.72, 7.64, 6.283, 4.975, 3.89, 2.959, 2.123, 1.552, 1.194,
00660     0.9949, 0.8938, 0.8268, 0.814, 0.8837, 0.3629, 0.449, 1.027, 1.698,
00661     2.339, 2.232, 1.709, 1.64, 1.636, 1.794, 1.865},
00662     {4.354, 4.279, 4.704, 5.365, 6.153, 7.049, 7.975, 8.793, 9.218,
00663     8.889, 7.805, 6.42, 5.093, 4.013, 3.112, 2.283, 1.671, 1.263,
00664     1.032, 0.9124, 0.8364, 0.8112, 0.8709, 0.9784, 0.5043, 0.9917, 1.587,
00665     2.354, 2.467, 1.85, 1.738, 1.724, 1.873, 1.918},
00666     {4.35, 4.331, 4.785, 5.473, 6.289, 7.199, 8.133, 8.898, 9.288,
00667     8.907, 7.796, 6.411, 5.112, 4.065, 3.212, 2.421, 1.788, 1.331,
00668     1.04, 0.8749, 0.769, 0.7242, 0.7799, 0.8189, 0.5724, 0.86, 1.151,
00669     1.587, 1.946, 1.722, 1.71, 1.845, 2.146, 2.307},
00670     {4.268, 4.264, 4.774, 5.45, 6.298, 7.255, 8.148, 8.907, 9.242,
00671     8.793, 7.612, 6.244, 5.029, 4.124, 3.411, 2.65, 1.98, 1.442,
00672     1.074, 0.8493, 0.7077, 0.6157, 0.5826, 0.5377, 0.4848, 0.7245, 0.8927,
00673     1.153, 1.565, 1.551, 1.638, 1.846, 2.153, 2.304},
00674     {4.119, 4.16, 4.652, 5.372, 6.224, 7.158, 8.06, 8.773, 9.034,
00675     8.507, 7.307, 5.989, 4.866, 4.046, 3.387, 2.637, 1.959, 1.402,
00676     1.01, 0.7722, 0.6266, 0.527, 0.462, 0.4124, 0.3452, 0.6418, 0.572,
00677     0.7352, 1.009, 1.245, 1.364, 1.556, 1.828, 1.914},
00678     {3.932, 4.041, 4.572, 5.285, 6.144, 7.099, 7.985, 8.671, 8.849,
00679     8.203, 6.936, 5.657, 4.621, 3.877, 3.266, 2.526, 1.845, 1.302,
00680     0.9214, 0.6906, 0.5486, 0.4553, 0.4002, 0.3533, 0.3033, 0.24, 0.4062,
00681     0.5084, 0.7133, 0.9299, 0.9714, 1.073, 1.204, 1.211},
00682     {3.882, 4.074, 4.658, 5.414, 6.269, 7.182, 8.05, 8.635, 8.639,
00683     7.828, 6.529, 5.303, 4.339, 3.626, 3.001, 2.262, 1.617, 1.128,
00684     0.7869, 0.574, 0.4488, 0.3713, 0.3216, 0.2794, 0.2403, 0.1962, 0.2638,
00685     0.3018, 0.4483, 0.5618, 0.5717, 0.5812, 0.5993, 0.5956},
00686     {4.195, 4.559, 5.287, 6.13, 7.024, 7.911, 8.623, 8.913, 8.545,
00687     7.44, 6.059, 4.92, 4.063, 3.391, 2.747, 1.953, 1.327, 0.9036,
00688     0.6235, 0.4566, 0.3534, 0.2935, 0.2547, 0.2226, 0.1871, 0.1494, 0.1959,
00689     0.1797, 0.2736, 0.3214, 0.3054, 0.2803, 0.2443, 0.1504},
00690     {4.644, 5.211, 6.192, 7.373, 8.534, 9.582, 10.35, 10.37, 9.49,
00691     7.907, 6.299, 5.088, 4.224, 3.492, 2.701, 1.795, 1.184, 0.8001,
00692     0.5348, 0.3789, 0.2861, 0.2435, 0.2055, 0.1754, 0.1605, 0.1076, 0.1644,
00693     0.09069, 0.1624, 0.2356, 0.223, 0.2001, 0.164, 0.09807}},
00694     {{0.0001362, 0.0001753, 0.0001792, 0.0001881, 0.0002233, 0.0002056,
00695     0.0001083, 1.505e-05, 1.073e-05,
00696     1.22e-05, 7.475e-06, 3.518e-06, 1.292e-06, 3.812e-07, 8.611e-08,
00697     3.148e-09, 1.29e-09, 2.111e-08,
00698     7.591e-07, 1.125e-06, 4.905e-06, 1.473e-05, 3.597e-05, 9.35e-05,
00699     0.0002623, 0.0001066, 0.0002064,
00700     0.0001924, 6.532e-10, 3.205e-10, 3.838e-10, 3.541e-10, 3.331e-10,
00701     2.774e-10}},
00702     {0.1865, 0.1757, 0.1596, 0.1454, 0.1343, 0.125, 0.1033, 0.08324, 0.06202,

```

00703 0.04341, 0.0295, 0.0187, 0.01129, 0.007115, 0.004932, 0.003824, 0.002733,
 00704 0.003019,
 00705 0.004022, 0.004428, 0.004414, 0.003826, 0.003009, 0.002117, 0.001375,
 00706 0.0009066, 0.001031,
 00707 0.0004673, 0.0002194, 0.0001683, 0.0001154, 6.607e-05, 3.128e-05,
 00708 1.601e-05},
 00709 {1.238, 1.384, 1.576, 1.788, 1.95, 1.993, 1.845, 1.593, 1.329,
 00710 1.098, 0.9387, 0.8174, 0.6531, 0.4579, 0.2899, 0.1818, 0.1253, 0.09461,
 00711 0.0741, 0.05946, 0.04989, 0.04353, 0.03862, 0.03469, 0.03181, 0.02862,
 00712 0.02499,
 00713 0.02062, 0.0178, 0.01681, 0.0162, 0.01679, 0.01921, 0.02192},
 00714 {1.92, 2.106, 2.478, 2.931, 3.421, 3.86, 4.043, 3.866, 3.41,
 00715 2.84, 2.372, 2.077, 1.825, 1.472, 1.07, 0.7023, 0.4688, 0.3296,
 00716 0.2453, 0.1889, 0.15, 0.1246, 0.1094, 0.09788, 0.09079, 0.08267, 0.08977,
 00717 0.09461, 0.09867, 0.09977, 0.09675, 0.09757, 0.1011, 0.1004},
 00718 {2.505, 2.661, 3.095, 3.659, 4.295, 4.928, 5.368, 5.413, 5.018,
 00719 4.291, 3.582, 3.052, 2.636, 2.193, 1.714, 1.199, 0.8121, 0.5616,
 00720 0.4193, 0.3284, 0.2656, 0.2245, 0.2009, 0.1819, 0.1615, 0.2142, 0.2142,
 00721 0.2523, 0.3029, 0.3253, 0.3248, 0.323, 0.3186, 0.3005},
 00722 {3.053, 3.153, 3.582, 4.16, 4.823, 5.525, 6.111, 6.38, 6.176,
 00723 5.453, 4.575, 3.84, 3.219, 2.652, 2.09, 1.509, 1.055, 0.7478,
 00724 0.5678, 0.4529, 0.3746, 0.322, 0.2979, 0.2602, 0.284, 0.3515, 0.39,
 00725 0.4915, 0.6421, 0.6745, 0.684, 0.7208, 0.7815, 0.8179},
 00726 {3.422, 3.44, 3.875, 4.477, 5.191, 5.965, 6.661, 7.123, 7.132,
 00727 6.48, 5.463, 4.526, 3.718, 3.005, 2.344, 1.688, 1.206, 0.8837,
 00728 0.6856, 0.5649, 0.4863, 0.445, 0.4856, 0.5006, 0.3136, 0.4515, 0.6109,
 00729 0.7863, 0.9684, 0.937, 0.937, 0.9949, 1.106, 1.204},
 00730 {3.782, 3.708, 4.085, 4.684, 5.42, 6.264, 7.105, 7.757, 7.965,
 00731 7.421, 6.309, 5.169, 4.161, 3.312, 2.538, 1.814, 1.314, 0.9921,
 00732 0.7986, 0.6906, 0.6241, 0.6037, 0.6883, 0.3135, 0.3539, 0.6781, 1.044,
 00733 1.41, 1.465, 1.198, 1.162, 1.195, 1.315, 1.399},
 00734 {4.093, 3.956, 4.303, 4.881, 5.64, 6.546, 7.473, 8.261, 8.617,
 00735 8.211, 7.109, 5.806, 4.611, 3.616, 2.718, 1.923, 1.396, 1.077,
 00736 0.8999, 0.7993, 0.7318, 0.69, 0.7736, 0.3307, 0.5444, 0.8973, 1.424,
 00737 1.867, 1.76, 1.423, 1.397, 1.429, 1.558, 1.631},
 00738 {4.316, 4.179, 4.572, 5.214, 6.033, 6.982, 7.93, 8.754, 9.154,
 00739 8.809, 7.719, 6.283, 4.962, 3.898, 2.961, 2.106, 1.526, 1.178,
 00740 0.9911, 0.8826, 0.8091, 0.792, 0.8984, 0.4081, 0.6591, 1.079, 1.762,
 00741 2.394, 2.262, 1.722, 1.649, 1.64, 1.793, 1.856},
 00742 {4.422, 4.322, 4.787, 5.478, 6.345, 7.323, 8.254, 9.118, 9.564,
 00743 9.218, 8.075, 6.604, 5.203, 4.058, 3.117, 2.274, 1.677, 1.284,
 00744 1.048, 0.9151, 0.8297, 0.7966, 0.8935, 0.5312, 0.7531, 1.072, 1.715,
 00745 2.495, 2.546, 1.897, 1.782, 1.77, 1.93, 1.983},
 00746 {4.547, 4.469, 4.942, 5.688, 6.549, 7.488, 8.486, 9.332, 9.775,
 00747 9.417, 8.264, 6.785, 5.379, 4.249, 3.33, 2.505, 1.852, 1.39,
 00748 1.098, 0.9308, 0.8263, 0.7895, 0.8576, 1.037, 0.635, 0.9803, 1.371,
 00749 1.867, 2.133, 1.823, 1.82, 1.997, 2.372, 2.584},
 00750 {4.687, 4.634, 5.104, 5.806, 6.687, 7.693, 8.664, 9.479, 9.89,
 00751 9.543, 8.412, 6.918, 5.518, 4.426, 3.541, 2.711, 2.014, 1.493,
 00752 1.144, 0.9317, 0.797, 0.7099, 0.6575, 0.6067, 0.526, 0.9154, 1.118,
 00753 1.507, 1.832, 1.788, 1.887, 2.179, 2.647, 2.868},
 00754 {4.695, 4.698, 5.185, 5.955, 6.871, 7.893, 8.877, 9.693, 10.1,
 00755 9.723, 8.528, 6.981, 5.591, 4.567, 3.788, 2.967, 2.242, 1.655,
 00756 1.222, 0.9406, 0.7599, 0.6428, 0.57, 0.513, 0.4496, 0.9249, 0.8597,
 00757 1.064, 1.428, 1.661, 1.831, 2.151, 2.551, 2.772},
 00758 {4.817, 4.823, 5.364, 6.154, 7.109, 8.194, 9.225, 10.09, 10.53,
 00759 10.13, 8.82, 7.204, 5.79, 4.774, 3.99, 3.147, 2.374, 1.734,
 00760 1.249, 0.9272, 0.7213, 0.5919, 0.522, 0.4691, 0.4205, 0.3435, 0.704,
 00761 0.795, 1.078, 1.369, 1.506, 1.697, 1.921, 2.057},
 00762 {5.178, 5.275, 5.904, 6.794, 7.806, 8.901, 9.941, 10.76, 11.06,
 00763 10.41, 8.899, 7.194, 5.78, 4.767, 3.992, 3.132, 2.304, 1.622,
 00764 1.125, 0.8041, 0.6135, 0.4964, 0.4334, 0.388, 0.3494, 0.3124, 0.5608,
 00765 0.6202, 0.8549, 1.142, 1.185, 1.327, 1.469, 1.521},
 00766 {5.765, 5.982, 6.854, 7.999, 9.258, 10.55, 11.75, 12.58, 12.65,
 00767 11.51, 9.567, 7.638, 6.122, 5.016, 4.17, 3.208, 2.283, 1.528,
 00768 0.9814, 0.6677, 0.4925, 0.3884, 0.3284, 0.2876, 0.2571, 0.2207, 0.5107,
 00769 0.5167, 0.7005, 0.8149, 0.7878, 0.7521, 0.6899, 0.4697},
 00770 {5.862, 6.014, 6.876, 8.048, 9.368, 10.8, 12.34, 13.65, 13.83,
 00771 12.41, 10.19, 8.106, 6.468, 5.285, 4.422, 3.392, 2.324, 1.509,
 00772 0.9796, 0.6834, 0.5019, 0.387, 0.3295, 0.2804, 0.2425, 0.2084, 0.4694,
 00773 0.4497, 0.5949, 0.6504, 0.6155, 0.5696, 0.5077, 0.2744},
 00774 {{6.617e-05, 8.467e-05, 8.509e-05, 9.824e-05, 0.0001317, 0.0001499,
 00775 0.0001104, 1.226e-05, 1.003e-05,
 00776 1.345e-05, 8.036e-06, 2.28e-06, 3.166e-07, 1.803e-08, 6.079e-10,
 00777 3.031e-10, 1.336e-09, 1.748e-08,
 00778 3.057e-07, 8.184e-07, 1.95e-06, 2.238e-06, 4.658e-06, 1.393e-05,
 00779 4.326e-05, 3.288e-05, 0.0001662,
 00780 8.012e-05, 6.48e-10, 3.371e-10, 4.509e-10, 4.052e-10, 3.677e-10,
 00781 2.996e-10},
 00782 {0.003936, 0.002158, 0.001688, 0.001555, 0.001396, 0.0008637, 0.0003091,
 00783 4.908e-05, 1.173e-05,
 00784 1.326e-05, 9.097e-06, 3.324e-06, 6.336e-07, 8.553e-08, 4.851e-09,
 00785 3.058e-09, 9.169e-09, 2.451e-07,
 00786 8.77e-07, 2.973e-07, 2.556e-07, 2.79e-07, 8.674e-07, 4.17e-06, 2.373e-05,
 00787 3.876e-05, 6.493e-05,
 00788 2.13e-05, 2.574e-09, 2.676e-10, 2.356e-10, 1.947e-10, 2.376e-10,
 00789 1.955e-10},


```

00790 {0.8642, 0.927, 0.9782, 1.001, 0.9705, 0.885, 0.7521, 0.6223, 0.5357,
00791 0.4614, 0.373, 0.2955, 0.2138, 0.1342, 0.07775, 0.04773, 0.0348, 0.02871,
00792 0.02503, 0.02329, 0.02308, 0.02227, 0.01967, 0.01687, 0.01381, 0.008063,
00793 0.006489,
00794 0.005433, 0.004804, 0.00487, 0.004735, 0.004696, 0.004587, 0.003634},
00795 {1.584, 1.731, 2.006, 2.368, 2.742, 3.047, 3.124, 2.894, 2.517,
00796 2.087, 1.761, 1.588, 1.399, 1.091, 0.7549, 0.4844, 0.3282, 0.2357,
00797 0.1737, 0.1299, 0.1001, 0.08162, 0.0699, 0.06056, 0.05333, 0.07014,
00798 0.04552,
00799 0.04907, 0.04781, 0.04938, 0.04864, 0.04971, 0.05141, 0.05078},
00800 {2.256, 2.4, 2.786, 3.287, 3.854, 4.41, 4.773, 4.776, 4.396,
00801 3.755, 3.16, 2.747, 2.381, 1.94, 1.458, 0.9865, 0.6576, 0.4582,
00802 0.3447, 0.2672, 0.2121, 0.1768, 0.1538, 0.1358, 0.1219, 0.1466, 0.1371,
00803 0.161, 0.1896, 0.2116, 0.2133, 0.2153, 0.2145, 0.2042},
00804 {2.844, 2.913, 3.296, 3.841, 4.473, 5.127, 5.672, 5.933, 5.724,
00805 5.005, 4.187, 3.549, 3.006, 2.486, 1.928, 1.368, 0.9363, 0.6524,
00806 0.4946, 0.3921, 0.3189, 0.2696, 0.2503, 0.2181, 0.2227, 0.2946, 0.297,
00807 0.3704, 0.4834, 0.5473, 0.5565, 0.581, 0.6189, 0.6392},
00808 {3.309, 3.337, 3.765, 4.35, 5.025, 5.756, 6.438, 6.908, 6.947,
00809 6.3, 5.252, 4.33, 3.576, 2.912, 2.258, 1.618, 1.15, 0.8271,
00810 0.6351, 0.5129, 0.4287, 0.3846, 0.4351, 0.4889, 0.3229, 0.4124, 0.4839,
00811 0.6285, 0.8468, 0.9091, 0.9059, 0.9413, 1.007, 1.074},
00812 {3.725, 3.662, 4.042, 4.647, 5.381, 6.204, 7.019, 7.636, 7.841,
00813 7.318, 6.177, 5.04, 4.09, 3.299, 2.545, 1.822, 1.303, 0.967,
00814 0.7683, 0.6506, 0.5816, 0.5712, 0.667, 0.3774, 0.3833, 0.5497, 0.7343,
00815 0.9899, 1.25, 1.213, 1.147, 1.144, 1.27, 1.389},
00816 {4.074, 3.916, 4.273, 4.851, 5.624, 6.564, 7.525, 8.296, 8.632,
00817 8.184, 7.013, 5.711, 4.593, 3.691, 2.813, 2.003, 1.445, 1.097,
00818 0.8982, 0.7815, 0.7103, 0.6929, 0.7841, 0.3697, 0.42, 0.8027, 1.255,
00819 1.716, 1.727, 1.411, 1.379, 1.416, 1.548, 1.618},
00820 {4.378, 4.233, 4.657, 5.297, 6.121, 7.098, 8.11, 8.937, 9.31,
00821 8.877, 7.696, 6.289, 5.029, 3.997, 3.066, 2.189, 1.589, 1.209,
00822 0.9953, 0.8642, 0.7883, 0.7739, 0.9011, 0.4115, 0.6312, 0.9904, 1.555,
00823 2.033, 1.915, 1.547, 1.523, 1.55, 1.675, 1.75},
00824 {4.574, 4.464, 4.871, 5.569, 6.471, 7.487, 8.486, 9.356, 9.788,
00825 9.387, 8.196, 6.717, 5.357, 4.245, 3.29, 2.387, 1.738, 1.316,
00826 1.064, 0.9101, 0.8224, 0.8068, 0.9022, 0.478, 0.7124, 1.093, 1.75,
00827 2.355, 2.233, 1.721, 1.663, 1.652, 1.836, 1.906},
00828 {4.771, 4.625, 5.041, 5.796, 6.698, 7.684, 8.72, 9.611, 10.08,
00829 9.736, 8.573, 7.013, 5.571, 4.447, 3.501, 2.606, 1.915, 1.439,
00830 1.139, 0.956, 0.8517, 0.8306, 0.9424, 1.177, 0.5869, 1.036, 1.44,
00831 1.878, 2.039, 1.705, 1.737, 1.923, 2.385, 2.641},
00832 {5.006, 4.866, 5.324, 6.09, 7.006, 8.036, 9.084, 9.893, 10.33,
00833 10.02, 8.839, 7.228, 5.736, 4.611, 3.71, 2.8, 2.063, 1.532,
00834 1.183, 0.9673, 0.8363, 0.7853, 0.8359, 0.8923, 0.569, 1.039, 1.365,
00835 1.742, 1.904, 1.758, 1.875, 2.251, 2.843, 3.224},
00836 {5.099, 4.961, 5.447, 6.211, 7.152, 8.22, 9.322, 10.2, 10.72,
00837 10.48, 9.248, 7.559, 5.993, 4.842, 3.975, 3.116, 2.35, 1.736,
00838 1.297, 1.003, 0.8134, 0.702, 0.6582, 0.6213, 0.5842, 0.5417, 1.158,
00839 1.4, 1.637, 1.863, 2.063, 2.497, 3.122, 3.523},
00840 {5.329, 5.217, 5.717, 6.57, 7.585, 8.724, 9.852, 10.78, 11.29,
00841 10.97, 9.671, 7.914, 6.3, 5.116, 4.263, 3.389, 2.58, 1.892,
00842 1.383, 1.036, 0.8131, 0.6799, 0.6149, 0.5698, 0.5283, 0.4671, 1.093,
00843 1.188, 1.482, 1.733, 1.908, 2.209, 2.612, 2.814},
00844 {6.071, 6.113, 6.75, 7.703, 8.826, 10.05, 11.16, 11.96, 12.25,
00845 11.61, 10.02, 8.102, 6.438, 5.24, 4.366, 3.448, 2.553, 1.795,
00846 1.245, 0.8829, 0.6737, 0.5561, 0.4996, 0.4579, 0.4257, 0.4016, 0.968,
00847 1.042, 1.284, 1.504, 1.639, 1.849, 2.098, 2.255},
00848 {6.417, 6.432, 7.23, 8.422, 9.846, 11.4, 12.82, 13.78, 14,
00849 12.98, 10.94, 8.756, 6.951, 5.69, 4.77, 3.716, 2.672, 1.818,
00850 1.224, 0.8517, 0.6496, 0.5355, 0.4766, 0.4254, 0.389, 0.351, 0.9729,
00851 0.9755, 1.108, 1.026, 0.9967, 0.9687, 0.9329, 0.8183},
00852 {6.462, 6.445, 7.221, 8.396, 9.796, 11.38, 12.96, 14.19, 14.59,
00853 13.52, 11.33, 9.046, 7.189, 5.92, 5.015, 3.949, 2.77, 1.841,
00854 1.239, 0.86, 0.6441, 0.5179, 0.446, 0.42, 0.3818, 0.3068, 0.9267,
00855 0.8849, 0.9828, 0.8458, 0.7544, 0.6963, 0.6233, 0.5358},
00856 {{3.24e-05, 4.086e-05, 4.221e-05, 5.381e-05, 8.808e-05, 0.0001261,
00857 0.0001263, 2.932e-05, 2.057e-05,
00858 4.321e-05, 3.43e-05, 1.213e-05, 1.862e-06, 1.067e-07, 2.983e-09,
00859 2.578e-08, 2.254e-09, 1.504e-08,
00860 1.599e-07, 1.523e-07, 2.033e-07, 5.271e-07, 1.417e-06, 4.518e-06,
00861 1.358e-05, 9.266e-06, 0.0001569,
00862 3.355e-05, 7.653e-10, 3.889e-10, 5.082e-10, 4.498e-10, 4.072e-10,
00863 3.302e-10},
00864 {0.0688, 0.05469, 0.04502, 0.03604, 0.02861, 0.02128, 0.01453, 0.00924,
00865 0.00574,
00866 0.003447, 0.001888, 0.001147, 0.001154, 0.001548, 0.001193, 6.955e-05,
00867 0.0003258, 1.673e-05,
00868 1.393e-05, 1.849e-05, 2.841e-05, 4.045e-05, 5.864e-05, 4.426e-05,
00869 1.722e-05, 3.186e-05, 5.582e-05,
00870 6.823e-06, 3.559e-09, 3.282e-10, 2.847e-10, 2.3e-10, 2.728e-10,
00871 2.237e-10},
00872 {1.056, 1.158, 1.28, 1.377, 1.429, 1.395, 1.242, 1.043, 0.8709,
00873 0.7435, 0.6183, 0.4824, 0.3665, 0.2668, 0.1749, 0.098, 0.09722, 0.08524,
00874 0.05545, 0.04501, 0.04476, 0.04491, 0.04231, 0.03868, 0.03336, 0.01108,
00875 0.01596,
00876 0.009172, 0.008499, 0.008763, 0.008672, 0.008872, 0.008926, 0.007552},

```

00877 {1.798, 1.952, 2.239, 2.589, 2.958, 3.261, 3.328, 3.12, 2.741,
 00878 2.293, 1.917, 1.686, 1.53, 1.28, 0.9362, 0.6052, 0.4093, 0.294,
 00879 0.2166, 0.1609, 0.1206, 0.09532, 0.08094, 0.07007, 0.06252, 0.1111,
 00880 0.04996,
 00881 0.0572, 0.0583, 0.06137, 0.06077, 0.0616, 0.0644, 0.06454},
 00882 {2.42, 2.566, 2.952, 3.45, 3.987, 4.499, 4.853, 4.878, 4.55,
 00883 3.918, 3.264, 2.802, 2.467, 2.094, 1.632, 1.127, 0.7523, 0.5168,
 00884 0.386, 0.2982, 0.2346, 0.1928, 0.1694, 0.1534, 0.1384, 0.1672, 0.1434,
 00885 0.1726, 0.2115, 0.2414, 0.2445, 0.2487, 0.2496, 0.2381},
 00886 {3.026, 3.096, 3.501, 4.016, 4.608, 5.244, 5.786, 6.088, 5.975,
 00887 5.326, 4.429, 3.701, 3.142, 2.636, 2.071, 1.477, 1.015, 0.7056,
 00888 0.5315, 0.4215, 0.3413, 0.2869, 0.2681, 0.2507, 0.2616, 0.3004, 0.3042,
 00889 0.3879, 0.5203, 0.6045, 0.6196, 0.651, 0.6986, 0.7171},
 00890 {3.463, 3.475, 3.878, 4.441, 5.112, 5.838, 6.522, 7.015, 7.096,
 00891 6.52, 5.487, 4.496, 3.734, 3.105, 2.456, 1.776, 1.26, 0.9016,
 00892 0.6841, 0.5485, 0.454, 0.404, 0.4609, 0.5432, 0.376, 0.4702, 0.5045,
 00893 0.6555, 0.9157, 1.076, 1.113, 1.216, 1.372, 1.497},
 00894 {3.745, 3.679, 4.071, 4.666, 5.394, 6.22, 7.064, 7.725, 7.961,
 00895 7.448, 6.321, 5.161, 4.233, 3.472, 2.73, 1.96, 1.409, 1.04,
 00896 0.8213, 0.6863, 0.6068, 0.5872, 0.7122, 0.9507, 0.4302, 0.5845, 0.768,
 00897 1.036, 1.308, 1.372, 1.393, 1.57, 2.023, 2.387},
 00898 {4.046, 3.927, 4.286, 4.869, 5.645, 6.599, 7.567, 8.4, 8.74,
 00899 8.247, 7.036, 5.735, 4.638, 3.741, 2.905, 2.066, 1.493, 1.132,
 00900 0.9307, 0.7993, 0.7306, 0.7334, 0.8811, 1.197, 0.4369, 0.7932, 1.239,
 00901 1.716, 1.764, 1.639, 1.738, 2.042, 2.543, 2.814},
 00902 {4.303, 4.169, 4.566, 5.224, 6.055, 7.053, 8.124, 9.037, 9.471,
 00903 8.998, 7.714, 6.277, 4.981, 3.935, 3.018, 2.182, 1.599, 1.224,
 00904 1.01, 0.8634, 0.7916, 0.7955, 0.9234, 1.237, 0.6362, 0.9588, 1.513,
 00905 1.996, 1.902, 1.545, 1.53, 1.572, 1.712, 1.768},
 00906 {4.506, 4.381, 4.819, 5.529, 6.428, 7.462, 8.518, 9.453, 9.907,
 00907 9.467, 8.237, 6.703, 5.29, 4.149, 3.213, 2.346, 1.732, 1.324,
 00908 1.075, 0.9181, 0.8397, 0.8375, 0.9434, 0.5018, 0.5414, 1.083, 1.74,
 00909 2.347, 2.227, 1.721, 1.661, 1.648, 1.831, 1.912},
 00910 {4.687, 4.558, 4.978, 5.725, 6.658, 7.731, 8.789, 9.675, 10.13,
 00911 9.748, 8.526, 6.958, 5.498, 4.351, 3.431, 2.537, 1.87, 1.415,
 00912 1.127, 0.9521, 0.8587, 0.8423, 0.8885, 1.184, 0.5938, 1.064, 1.469,
 00913 1.877, 2.006, 1.669, 1.7, 1.879, 2.347, 2.626},
 00914 {4.936, 4.809, 5.237, 6.008, 6.937, 7.97, 9.016, 9.924, 10.41,
 00915 10.04, 8.758, 7.153, 5.667, 4.529, 3.633, 2.737, 2.011, 1.493,
 00916 1.161, 0.9601, 0.8439, 0.8078, 0.8976, 1.023, 0.6551, 1.239, 1.595,
 00917 1.9, 1.923, 1.721, 1.833, 2.227, 2.857, 3.283},
 00918 {5.075, 4.962, 5.428, 6.241, 7.2, 8.253, 9.328, 10.2, 10.65,
 00919 10.28, 8.983, 7.308, 5.772, 4.63, 3.752, 2.906, 2.151, 1.57,
 00920 1.188, 0.9525, 0.8028, 0.7168, 0.7021, 0.7201, 0.7339, 0.6036, 1.423,
 00921 1.574, 1.715, 1.841, 2.05, 2.525, 3.214, 3.663},
 00922 {5.33, 5.234, 5.731, 6.529, 7.518, 8.638, 9.699, 10.59, 11.06,
 00923 10.69, 9.363, 7.602, 6.028, 4.875, 4.029, 3.2, 2.426, 1.77,
 00924 1.305, 0.9937, 0.7977, 0.6831, 0.6247, 0.6058, 0.5859, 0.5171, 1.306,
 00925 1.309, 1.552, 1.652, 1.83, 2.179, 2.619, 2.844},
 00926 {5.866, 5.848, 6.455, 7.346, 8.396, 9.533, 10.53, 11.34, 11.68,
 00927 11.13, 9.619, 7.806, 6.227, 5.073, 4.235, 3.4, 2.554, 1.805,
 00928 1.264, 0.9221, 0.7291, 0.623, 0.5765, 0.5499, 0.5313, 0.5058, 1.13,
 00929 1.193, 1.384, 1.521, 1.658, 1.903, 2.204, 2.394},
 00930 {6.303, 6.398, 7.222, 8.351, 9.705, 11.15, 12.37, 13.21, 13.37,
 00931 12.4, 10.47, 8.393, 6.679, 5.472, 4.597, 3.653, 2.67, 1.827,
 00932 1.234, 0.8789, 0.697, 0.5977, 0.5427, 0.5065, 0.4765, 0.4416, 1.108,
 00933 1.122, 1.2, 1.026, 0.9995, 0.9762, 0.9406, 0.8155},
 00934 {6.374, 6.361, 7.22, 8.387, 9.771, 11.27, 12.69, 13.72, 14.07,
 00935 13.07, 10.88, 8.681, 6.947, 5.79, 4.913, 3.885, 2.773, 1.855,
 00936 1.244, 0.9142, 0.7256, 0.598, 0.5151, 0.5035, 0.4732, 0.3585, 1.059,
 00937 1.045, 1.089, 0.8483, 0.7697, 0.7077, 0.6288, 0.5436},
 00938 {{0.03789, 0.03408, 0.03134, 0.02846, 0.02395, 0.01876, 0.01296, 0.008683,
 00939 0.005595,
 00940 0.003327, 0.001899, 0.001016, 0.0006645, 0.001147, 0.000686, 0.0006987,
 00941 0.0006744, 0.0003903,
 00942 1.823e-05, 1.418e-05, 1.097e-05, 9.197e-06, 8.385e-06, 5.361e-06,
 00943 1.047e-05, 8.724e-06, 0.0001224,
 00944 1.608e-05, 7.761e-10, 4.005e-10, 5.216e-10, 4.653e-10, 4.282e-10,
 00945 3.472e-10},
 00946 {0.9685, 1.035, 1.103, 1.161, 1.165, 1.106, 0.9814, 0.8435, 0.7222,
 00947 0.6291, 0.5477, 0.4518, 0.3058, 0.2105, 0.1492, 0.08369, 0.1841, 0.1379,
 00948 0.1032, 0.06085, 0.03194, 0.02288, 0.0206, 0.02385, 0.0298, 0.0102,
 00949 0.002722,
 00950 0.00743, 0.007294, 0.006871, 0.006275, 0.005435, 0.004153, 0.002587},
 00951 {1.842, 2.002, 2.295, 2.618, 2.95, 3.205, 3.226, 2.966, 2.55,
 00952 2.078, 1.668, 1.355, 1.079, 0.8142, 0.7131, 0.5158, 0.3251, 0.2412,
 00953 0.1924, 0.1651, 0.1599, 0.164, 0.1325, 0.1215, 0.09591, 0.1117, 0.03993,
 00954 0.04814, 0.04778, 0.04729, 0.04627, 0.0467, 0.04597, 0.04172},
 00955 {2.489, 2.643, 3.032, 3.507, 4.035, 4.533, 4.829, 4.772, 4.375,
 00956 3.714, 3.06, 2.551, 2.207, 1.938, 1.628, 1.185, 0.8109, 0.5625,
 00957 0.4101, 0.3078, 0.2325, 0.1813, 0.1507, 0.1311, 0.1178, 0.1964, 0.1063,
 00958 0.1261, 0.1425, 0.1561, 0.1546, 0.1532, 0.1543, 0.1498},
 00959 {3.03, 3.156, 3.579, 4.144, 4.775, 5.42, 5.935, 6.129, 5.878,
 00960 5.138, 4.235, 3.481, 2.956, 2.542, 2.086, 1.502, 1.028, 0.7057,
 00961 0.5223, 0.4097, 0.3291, 0.2729, 0.2417, 0.2235, 0.2035, 0.2565, 0.2228,
 00962 0.274, 0.339, 0.387, 0.3918, 0.396, 0.3931, 0.3692},
 00963 {3.447, 3.508, 3.934, 4.533, 5.226, 5.96, 6.611, 7.003, 6.968,

```

00964      6.331, 5.301, 4.324, 3.576, 3.009, 2.45, 1.808, 1.285, 0.894,
00965      0.6612, 0.5244, 0.4273, 0.3588, 0.3341, 0.3096, 0.296, 0.4036, 0.403,
00966      0.5114, 0.6763, 0.7735, 0.7969, 0.8461, 0.9229, 0.9621},
00967 {3.722, 3.706, 4.103, 4.704, 5.429, 6.225, 6.997, 7.531, 7.688,
00968      7.197, 6.151, 5.021, 4.119, 3.436, 2.786, 2.084, 1.513, 1.082,
00969      0.808, 0.6428, 0.5308, 0.473, 0.5475, 0.6436, 0.4784, 0.5735, 0.5953,
00970      0.7959, 1.106, 1.287, 1.339, 1.478, 1.689, 1.86},
00971 {3.921, 3.841, 4.189, 4.783, 5.529, 6.4, 7.263, 7.978, 8.293,
00972      7.875, 6.788, 5.549, 4.498, 3.668, 2.902, 2.117, 1.551, 1.158,
00973      0.9124, 0.7546, 0.6628, 0.6526, 0.7867, 0.9666, 0.4912, 0.664, 0.8688,
00974      1.172, 1.475, 1.537, 1.57, 1.791, 2.33, 2.77},
00975 {4.115, 3.995, 4.345, 4.915, 5.667, 6.583, 7.547, 8.4, 8.806,
00976      8.416, 7.274, 5.963, 4.777, 3.806, 2.936, 2.114, 1.555, 1.194,
00977      0.9788, 0.8251, 0.7444, 0.7323, 0.8067, 0.4073, 0.4371, 0.8223, 1.294,
00978      1.799, 1.845, 1.715, 1.83, 2.164, 2.718, 3.019},
00979 {4.267, 4.151, 4.545, 5.138, 5.928, 6.9, 7.906, 8.818, 9.276,
00980      8.888, 7.7, 6.275, 4.923, 3.83, 2.949, 2.126, 1.57, 1.211,
00981      0.9919, 0.8349, 0.7611, 0.7613, 0.8757, 0.4158, 0.6205, 0.9285, 1.474,
00982      1.966, 1.886, 1.546, 1.534, 1.58, 1.727, 1.781},
00983 {4.357, 4.249, 4.661, 5.37, 6.245, 7.245, 8.255, 9.142, 9.591,
00984      9.187, 7.974, 6.489, 5.106, 3.983, 3.055, 2.223, 1.646, 1.263,
00985      1.021, 0.8595, 0.7835, 0.7906, 0.9308, 0.4752, 0.6857, 1.015, 1.644,
00986      2.238, 2.142, 1.682, 1.628, 1.619, 1.797, 1.869},
00987 {4.506, 4.431, 4.893, 5.628, 6.535, 7.541, 8.516, 9.354, 9.751,
00988      9.305, 8.052, 6.578, 5.209, 4.113, 3.227, 2.379, 1.745, 1.32,
00989      1.053, 0.8816, 0.796, 0.7899, 0.9061, 1.191, 0.5672, 0.9669, 1.349,
00990      1.746, 1.888, 1.592, 1.62, 1.791, 2.222, 2.485},
00991 {4.576, 4.506, 4.983, 5.745, 6.664, 7.67, 8.657, 9.446, 9.792,
00992      9.306, 8.032, 6.569, 5.251, 4.219, 3.374, 2.515, 1.835, 1.359,
00993      1.057, 0.8733, 0.7654, 0.7338, 0.8346, 1.067, 0.6038, 1.054, 1.38,
00994      1.684, 1.756, 1.589, 1.679, 2.018, 2.626, 2.986},
00995 {4.675, 4.625, 5.078, 5.828, 6.714, 7.661, 8.588, 9.4, 9.761,
00996      9.26, 7.973, 6.455, 5.133, 4.127, 3.339, 2.562, 1.884, 1.37,
00997      1.036, 0.8316, 0.7014, 0.6314, 0.6329, 0.6673, 0.7434, 1.216, 1.142,
00998      1.346, 1.515, 1.614, 1.773, 2.186, 2.758, 3.13},
00999 {4.664, 4.649, 5.144, 5.923, 6.826, 7.78, 8.69, 9.369, 9.701,
01000      9.297, 8.02, 6.479, 5.147, 4.209, 3.474, 2.764, 2.093, 1.519,
01001      1.113, 0.8486, 0.683, 0.5849, 0.5324, 0.5209, 0.5327, 1.058, 0.9241,
01002      1.048, 1.287, 1.427, 1.546, 1.782, 2.111, 2.301},
01003 {4.771, 4.822, 5.408, 6.232, 7.155, 8.095, 8.92, 9.55, 9.74,
01004      9.14, 7.776, 6.287, 5.065, 4.198, 3.518, 2.824, 2.145, 1.537,
01005      1.096, 0.8121, 0.644, 0.5446, 0.4895, 0.4683, 0.463, 0.4501, 0.773,
01006      0.7994, 0.9657, 1.138, 1.222, 1.365, 1.564, 1.666},
01007 {5.238, 5.493, 6.287, 7.287, 8.326, 9.282, 10.02, 10.41, 10.22,
01008      9.194, 7.64, 6.172, 5.058, 4.272, 3.57, 2.766, 1.962, 1.313,
01009      0.8882, 0.6342, 0.4953, 0.4159, 0.3845, 0.3668, 0.3524, 0.3383, 0.6226,
01010      0.6524, 0.702, 0.662, 0.6453, 0.6304, 0.6111, 0.5462},
01011 {5.459, 5.786, 6.717, 8.018, 9.426, 10.72, 11.67, 11.98, 11.43,
01012      9.923, 8.082, 6.51, 5.425, 4.719, 3.958, 2.937, 1.953, 1.26,
01013      0.8432, 0.5868, 0.4389, 0.3576, 0.3281, 0.3182, 0.296, 0.2263, 0.5208,
01014      0.5264, 0.5272, 0.439, 0.4076, 0.3774, 0.3392, 0.2954}},
01015 {{1.93, 2.082, 2.236, 2.401, 2.486, 2.46, 2.242, 1.936, 1.632,
01016      1.309, 1.205, 0.996, 0.8843, 0.5832, 0.3788, 0.2472, 0.1935, 0.199,
01017      0.2177, 0.3668, 0.2468, 0.1727, 0.1235, 0.1211, 0.09577, 0.05738, 0.01593,
01018      0.01572, 0.01585, 0.01477, 0.01213, 0.01077, 0.008684, 0.005934},
01019      {2.406, 2.637, 3.006, 3.467, 3.941, 4.339, 4.464, 4.221, 3.692,
01020      2.961, 2.333, 1.856, 1.579, 1.321, 0.9877, 0.7954, 0.535, 0.3953,
01021      0.3269, 0.3153, 0.4016, 0.4948, 0.4946, 0.3969, 0.2986, 0.157, 0.08327,
01022      0.09294, 0.1047, 0.09143, 0.08129, 0.06982, 0.05108, 0.0324},
01023      {2.891, 3.082, 3.531, 4.114, 4.759, 5.387, 5.81, 5.856, 5.447,
01024      4.602, 3.716, 2.967, 2.422, 1.997, 1.543, 1.312, 1.043, 0.7202,
01025      0.5009, 0.3828, 0.3369, 0.3204, 0.3053, 0.2956, 0.2344, 0.3256, 0.2033,
01026      0.2183, 0.2574, 0.252, 0.2454, 0.2496, 0.2494, 0.2289},
01027      {3.257, 3.412, 3.896, 4.558, 5.307, 6.077, 6.732, 7.047, 6.849,
01028      6.037, 4.935, 3.973, 3.242, 2.755, 2.32, 1.807, 1.313, 0.9215,
01029      0.6619, 0.4999, 0.3902, 0.3134, 0.2686, 0.245, 0.2368, 0.2048, 0.1813,
01030      0.2732, 0.3298, 0.3568, 0.3526, 0.3493, 0.3549, 0.3469},
01031      {3.587, 3.682, 4.173, 4.839, 5.622, 6.472, 7.24, 7.722, 7.706,
01032      6.99, 5.826, 4.706, 3.861, 3.255, 2.675, 1.997, 1.417, 0.9866,
01033      0.7187, 0.561, 0.4546, 0.3832, 0.3527, 0.3395, 0.3279, 0.4213, 0.3649,
01034      0.4532, 0.5745, 0.6214, 0.6234, 0.6274, 0.6214, 0.5786},
01035      {3.86, 3.88, 4.313, 4.965, 5.741, 6.605, 7.439, 8.066, 8.231,
01036      7.669, 6.514, 5.29, 4.308, 3.595, 2.953, 2.22, 1.613, 1.142,
01037      0.8371, 0.6574, 0.5366, 0.4574, 0.4295, 0.4016, 0.3794, 0.5616, 0.5829,
01038      0.7168, 0.9521, 1.025, 1.053, 1.129, 1.25, 1.317},
01039      {4.081, 4.041, 4.427, 5.032, 5.788, 6.668, 7.53, 8.225, 8.513,
01040      8.083, 6.997, 5.729, 4.642, 3.826, 3.135, 2.369, 1.741, 1.266,
01041      0.9467, 0.7487, 0.6194, 0.5505, 0.6097, 0.7323, 0.5351, 0.6646, 0.7098,
01042      0.9842, 1.318, 1.49, 1.534, 1.718, 2.035, 2.285},
01043      {4.216, 4.126, 4.503, 5.084, 5.827, 6.712, 7.64, 8.42, 8.793,
01044      8.425, 7.34, 6.028, 4.838, 3.895, 3.074, 2.264, 1.681, 1.272,
01045      1.009, 0.8308, 0.7345, 0.7299, 0.8734, 1.142, 0.5401, 0.7387, 0.9717,
01046      1.328, 1.652, 1.667, 1.69, 1.954, 2.622, 3.19},
01047      {4.272, 4.151, 4.513, 5.089, 5.851, 6.773, 7.683, 8.504, 8.944,
01048      8.636, 7.572, 6.215, 4.931, 3.883, 2.974, 2.147, 1.591, 1.231,
01049      1.009, 0.859, 0.7919, 0.8024, 0.896, 0.4226, 0.5516, 0.8571, 1.354,
01050      1.883, 1.918, 1.778, 1.904, 2.264, 2.865, 3.19},

```

01051 {4.268, 4.149, 4.512, 5.115, 5.895, 6.826, 7.773, 8.616, 9.068,
 01052 8.77, 7.691, 6.307, 4.976, 3.859, 2.932, 2.123, 1.578, 1.222,
 01053 0.9995, 0.8488, 0.7895, 0.8146, 0.9335, 0.4125, 0.6071, 0.932, 1.493,
 01054 1.986, 1.896, 1.55, 1.538, 1.583, 1.731, 1.792},
 01055 {4.24, 4.152, 4.565, 5.21, 6.012, 6.947, 7.876, 8.687, 9.116,
 01056 8.769, 7.647, 6.266, 4.953, 3.871, 2.972, 2.151, 1.594, 1.22,
 01057 0.9831, 0.8291, 0.7694, 0.7899, 0.9141, 0.4431, 0.6319, 0.9495, 1.544,
 01058 2.116, 2.045, 1.603, 1.547, 1.538, 1.698, 1.778},
 01059 {4.22, 4.152, 4.587, 5.265, 6.061, 6.961, 7.879, 8.672, 9.045,
 01060 8.614, 7.444, 6.083, 4.845, 3.844, 2.996, 2.197, 1.608, 1.21,
 01061 0.9591, 0.8021, 0.7294, 0.7305, 0.8376, 1.102, 0.495, 0.808, 1.134,
 01062 1.49, 1.689, 1.458, 1.475, 1.619, 1.978, 2.214},
 01063 {4.139, 4.115, 4.593, 5.262, 6.061, 6.95, 7.815, 8.557, 8.84,
 01064 8.311, 7.1, 5.777, 4.638, 3.742, 2.972, 2.201, 1.594, 1.167,
 01065 0.9052, 0.7436, 0.6504, 0.62, 0.7047, 0.8275, 0.483, 0.7578, 0.9831,
 01066 1.246, 1.432, 1.377, 1.433, 1.677, 2.116, 2.395},
 01067 {4.027, 4.03, 4.521, 5.195, 5.986, 6.839, 7.642, 8.241, 8.387,
 01068 7.78, 6.581, 5.331, 4.311, 3.538, 2.878, 2.173, 1.57, 1.118,
 01069 0.8368, 0.6691, 0.5655, 0.5116, 0.5213, 0.5546, 0.5962, 0.7812, 0.7265,
 01070 0.8823, 1.107, 1.292, 1.385, 1.633, 2.016, 2.214},
 01071 {3.787, 3.852, 4.369, 5.07, 5.845, 6.625, 7.28, 7.735, 7.755,
 01072 7.082, 5.9, 4.755, 3.857, 3.202, 2.64, 2.031, 1.478, 1.036,
 01073 0.7546, 0.5863, 0.4836, 0.422, 0.3872, 0.3827, 0.4058, 0.6138, 0.5067,
 01074 0.6048, 0.7857, 0.9735, 1.052, 1.172, 1.322, 1.358},
 01075 {3.556, 3.717, 4.24, 4.935, 5.651, 6.309, 6.815, 7.1, 6.959,
 01076 6.199, 5.099, 4.12, 3.39, 2.875, 2.37, 1.798, 1.291, 0.8979,
 01077 0.6433, 0.4909, 0.3991, 0.3437, 0.3063, 0.2936, 0.2983, 0.283, 0.3393,
 01078 0.3825, 0.4825, 0.6185, 0.6697, 0.7099, 0.7581, 0.7492},
 01079 {3.344, 3.611, 4.169, 4.812, 5.45, 5.983, 6.257, 6.263, 5.844,
 01080 4.989, 4.049, 3.337, 2.825, 2.403, 1.893, 1.346, 0.9248, 0.6385,
 01081 0.4618, 0.3493, 0.281, 0.2397, 0.2114, 0.1989, 0.1936, 0.1739, 0.1941,
 01082 0.2137, 0.2382, 0.2613, 0.2565, 0.2526, 0.2392, 0.2172},
 01083 {3.617, 4.108, 4.785, 5.409, 5.873, 6.058, 5.747, 5.269, 4.577,
 01084 3.776, 3.142, 2.721, 2.304, 1.764, 1.195, 0.7421, 0.4648, 0.3052,
 01085 0.2132, 0.1552, 0.1201, 0.1028, 0.09496, 0.09272, 0.092, 0.06578, 0.09663,
 01086 0.1039, 0.1161, 0.1127, 0.1081, 0.1019, 0.09228, 0.08144}},
 01087 {{4.776, 5.263, 6.105, 7.209, 8.284, 9.138, 9.553, 9.22, 8.161,
 01088 6.644, 5.129, 4.002, 3.207, 2.842, 2.466, 1.792, 1.171, 0.7381,
 01089 0.5022, 0.4253, 0.4401, 0.5454, 0.7785, 0.6418, 0.4899, 0.3887, 0.3859,
 01090 0.2463, 0.248, 0.1831, 0.1414, 0.1233, 0.104, 0.07015},
 01091 {4.315, 4.602, 5.253, 6.073, 6.933, 7.732, 8.306, 8.428, 7.938,
 01092 6.819, 5.489, 4.358, 3.446, 2.895, 2.476, 1.885, 1.473, 1.07,
 01093 0.7373, 0.5219, 0.4618, 0.454, 0.4569, 0.4263, 0.397, 0.4683, 0.4728,
 01094 0.4348, 0.4638, 0.3711, 0.3221, 0.2786, 0.2347, 0.153},
 01095 {4.034, 4.197, 4.772, 5.576, 6.46, 7.358, 8.131, 8.568, 8.457,
 01096 7.592, 6.31, 5.095, 4.065, 3.363, 2.711, 2.026, 1.686, 1.33,
 01097 1.006, 0.7144, 0.5379, 0.4588, 0.4427, 0.451, 0.4738, 0.5938, 0.595,
 01098 0.5555, 0.6354, 0.6081, 0.5877, 0.5953, 0.6217, 0.6394},
 01099 {4.052, 4.154, 4.699, 5.472, 6.377, 7.353, 8.242, 8.879, 9.016,
 01100 8.382, 7.119, 5.777, 4.673, 3.861, 3.197, 2.408, 1.717, 1.19,
 01101 0.8643, 0.6598, 0.5193, 0.4314, 0.3937, 0.3863, 0.4011, 0.3591, 0.4105,
 01102 0.4673, 0.5539, 0.5717, 0.5531, 0.5518, 0.5563, 0.5379},
 01103 {4.153, 4.213, 4.735, 5.466, 6.334, 7.3, 8.25, 8.998, 9.286,
 01104 8.779, 7.558, 6.165, 5, 4.154, 3.419, 2.575, 1.849, 1.298,
 01105 0.9519, 0.7439, 0.6103, 0.537, 0.545, 0.5816, 0.5683, 0.7049, 0.6064,
 01106 0.7504, 0.9181, 0.9021, 0.8857, 0.8707, 0.8466, 0.7728},
 01107 {4.248, 4.254, 4.738, 5.442, 6.269, 7.19, 8.14, 8.939, 9.34,
 01108 8.978, 7.82, 6.411, 5.184, 4.273, 3.525, 2.664, 1.942, 1.394,
 01109 1.033, 0.8143, 0.6732, 0.5954, 0.6154, 0.6352, 0.7198, 0.8339, 0.8119,
 01110 1.046, 1.311, 1.244, 1.264, 1.359, 1.508, 1.629},
 01111 {4.354, 4.315, 4.761, 5.42, 6.219, 7.126, 8.036, 8.844, 9.254,
 01112 8.926, 7.833, 6.444, 5.174, 4.22, 3.457, 2.639, 1.948, 1.432,
 01113 1.087, 0.8685, 0.7349, 0.6908, 0.8, 0.9447, 0.6798, 0.8977, 1.108,
 01114 1.425, 1.666, 1.47, 1.449, 1.545, 1.768, 1.943},
 01115 {4.362, 4.274, 4.672, 5.312, 6.096, 7.012, 7.964, 8.785, 9.19,
 01116 8.837, 7.774, 6.356, 5.026, 3.988, 3.137, 2.353, 1.747, 1.332,
 01117 1.065, 0.8964, 0.8178, 0.823, 0.9551, 1.158, 0.5854, 0.9884, 1.483,
 01118 2.035, 2.091, 1.603, 1.521, 1.55, 1.723, 1.854},
 01119 {4.3, 4.174, 4.537, 5.167, 5.965, 6.89, 7.821, 8.663, 9.083,
 01120 8.735, 7.634, 6.25, 4.936, 3.878, 2.975, 2.165, 1.611, 1.244,
 01121 1.01, 0.864, 0.8216, 0.8541, 0.947, 0.4242, 0.4811, 0.9771, 1.569,
 01122 2.131, 2.042, 1.616, 1.566, 1.597, 1.77, 1.859},
 01123 {4.191, 4.082, 4.474, 5.086, 5.853, 6.754, 7.678, 8.491, 8.887,
 01124 8.503, 7.408, 6.052, 4.775, 3.739, 2.846, 2.066, 1.531, 1.182,
 01125 0.9574, 0.8217, 0.7847, 0.8129, 0.9004, 0.3845, 0.5912, 0.9488, 1.54,
 01126 2.073, 1.987, 1.591, 1.544, 1.55, 1.68, 1.746},
 01127 {4.067, 4, 4.384, 4.981, 5.736, 6.618, 7.486, 8.26, 8.602,
 01128 8.169, 7.063, 5.748, 4.553, 3.579, 2.743, 1.997, 1.466, 1.117,
 01129 0.8947, 0.756, 0.7085, 0.7246, 0.8226, 0.3804, 0.4137, 0.8378, 1.338,
 01130 1.831, 1.823, 1.452, 1.396, 1.406, 1.546, 1.607},
 01131 {3.852, 3.824, 4.228, 4.825, 5.559, 6.404, 7.233, 7.922, 8.179,
 01132 7.678, 6.56, 5.348, 4.269, 3.383, 2.62, 1.89, 1.369, 1.029,
 01133 0.8128, 0.6781, 0.613, 0.6018, 0.6705, 0.6956, 0.4126, 0.6338, 0.8946,
 01134 1.186, 1.405, 1.278, 1.277, 1.398, 1.661, 1.83},
 01135 {3.577, 3.6, 4.019, 4.632, 5.336, 6.114, 6.899, 7.485, 7.599,
 01136 6.995, 5.898, 4.813, 3.907, 3.169, 2.502, 1.82, 1.298, 0.9403,
 01137 0.7213, 0.5834, 0.4987, 0.4571, 0.4767, 0.4929, 0.357, 0.5342, 0.6218,

```

01138     0.8321, 1.072, 1.133, 1.182, 1.326, 1.548, 1.687},
01139     {3.251, 3.338, 3.79, 4.396, 5.094, 5.833, 6.465, 6.856, 6.763,
01140     6.045, 5.018, 4.111, 3.394, 2.814, 2.265, 1.671, 1.175, 0.8195,
01141     0.6074, 0.4777, 0.3953, 0.3477, 0.3295, 0.3281, 0.3202, 0.3996, 0.4085,
01142     0.5264, 0.6945, 0.8446, 0.9062, 1.002, 1.145, 1.182},
01143     {2.844, 3.014, 3.474, 4.053, 4.689, 5.317, 5.776, 5.907, 5.606,
01144     4.864, 3.98, 3.263, 2.726, 2.292, 1.824, 1.308, 0.8985, 0.62,
01145     0.4532, 0.3508, 0.2879, 0.2496, 0.2268, 0.217, 0.2089, 0.1723, 0.2384,
01146     0.3033, 0.3816, 0.457, 0.4738, 0.4945, 0.522, 0.511},
01147     {2.32, 2.558, 2.981, 3.486, 3.981, 4.383, 4.505, 4.35, 3.92,
01148     3.298, 2.688, 2.238, 1.901, 1.574, 1.197, 0.8142, 0.5528, 0.3893,
01149     0.2874, 0.2207, 0.179, 0.153, 0.1353, 0.126, 0.118, 0.0996, 0.1038,
01150     0.1254, 0.1556, 0.1673, 0.1637, 0.1606, 0.1599, 0.1512},
01151     {1.604, 1.812, 2.085, 2.33, 2.517, 2.565, 2.341, 2.07, 1.768,
01152     1.479, 1.254, 1.072, 0.8657, 0.6304, 0.4269, 0.2773, 0.1887, 0.1365,
01153     0.104, 0.08106, 0.06679, 0.05742, 0.0504, 0.04538, 0.04064, 0.03211,
01154     0.02611,
01155     0.02727, 0.03066, 0.03125, 0.02913, 0.02667, 0.02485, 0.02217},
01156     {0.4429, 0.4652, 0.4579, 0.4357, 0.388, 0.3401, 0.2901, 0.2551, 0.2192,
01157     0.1788, 0.1348, 0.0914, 0.05502, 0.03148, 0.01858, 0.01216, 0.009078,
01158     0.007534,
01159     0.006492, 0.006257, 0.006301, 0.006115, 0.005378, 0.005084, 0.005022,
01160     0.002766, 0.00246,
01161     0.001431, 0.001208, 0.0014, 0.001257, 0.001091, 0.0009076, 0.0005632}},
01162     {{6.159, 6.267, 7.137, 8.441, 9.868, 11.32, 12.68, 13.44, 13.31,
01163     11.97, 9.899, 7.982, 6.529, 5.456, 4.582, 3.411, 2.447, 1.735,
01164     1.201, 0.8902, 0.7385, 0.82, 0.8247, 0.6792, 0.5978, 0.7594, 1.311,
01165     0.8001, 0.7222, 0.5196, 0.4, 0.3513, 0.3049, 0.2033},
01166     {6.025, 6.218, 7.108, 8.305, 9.599, 10.86, 11.89, 12.5, 12.4,
01167     11.24, 9.372, 7.562, 6.173, 5.198, 4.359, 3.187, 2.293, 1.674,
01168     1.209, 0.8758, 0.6848, 0.7086, 0.7544, 0.6892, 0.6462, 0.8934, 1.25,
01169     0.9268, 0.9078, 0.7087, 0.6172, 0.5399, 0.4671, 0.304},
01170     {5.423, 5.508, 6.112, 6.997, 8.017, 9.107, 10.07, 10.75, 10.95,
01171     10.26, 8.775, 7.139, 5.746, 4.723, 3.902, 2.898, 2.14, 1.579,
01172     1.155, 0.86, 0.6747, 0.5805, 0.5689, 0.5958, 0.6918, 0.8863, 1.028,
01173     0.8369, 0.8899, 0.8223, 0.7867, 0.7493, 0.7297, 0.6911},
01174     {5.03, 5.007, 5.533, 6.337, 7.269, 8.306, 9.353, 10.16, 10.53,
01175     10.09, 8.802, 7.176, 5.772, 4.785, 3.999, 3.116, 2.292, 1.613,
01176     1.161, 0.8825, 0.7044, 0.602, 0.5675, 0.5799, 0.633, 0.5335, 0.75,
01177     0.782, 0.8887, 0.8566, 0.8053, 0.779, 0.7512, 0.6894},
01178     {4.854, 4.795, 5.239, 5.991, 6.89, 7.916, 8.949, 9.794, 10.25,
01179     9.932, 8.787, 7.214, 5.806, 4.824, 4.046, 3.151, 2.318, 1.656,
01180     1.216, 0.9337, 0.7505, 0.6429, 0.6093, 0.6241, 0.6313, 0.828, 0.8414,
01181     1.007, 1.191, 1.083, 1.037, 1.002, 0.963, 0.8665},
01182     {4.689, 4.616, 5.08, 5.801, 6.661, 7.626, 8.619, 9.471, 9.972,
01183     9.711, 8.582, 7.039, 5.65, 4.664, 3.923, 3.04, 2.236, 1.63,
01184     1.223, 0.9546, 0.7831, 0.6819, 0.6534, 0.6599, 0.5823, 0.9021, 1.016,
01185     1.267, 1.495, 1.395, 1.398, 1.508, 1.711, 1.859},
01186     {4.575, 4.466, 4.925, 5.634, 6.479, 7.412, 8.343, 9.218, 9.74,
01187     9.468, 8.319, 6.801, 5.4, 4.37, 3.571, 2.713, 2.014, 1.498,
01188     1.158, 0.9364, 0.7954, 0.7273, 0.7681, 0.8566, 0.6687, 0.9463, 1.219,
01189     1.575, 1.788, 1.546, 1.525, 1.635, 1.887, 2.09},
01190     {4.408, 4.299, 4.735, 5.433, 6.275, 7.22, 8.175, 9.032, 9.47,
01191     9.097, 7.942, 6.45, 5.08, 4.019, 3.16, 2.338, 1.735, 1.324,
01192     1.063, 0.9095, 0.8421, 0.8249, 0.8269, 0.6041, 0.6079, 1.07, 1.601,
01193     2.169, 2.204, 1.68, 1.593, 1.624, 1.805, 1.939},
01194     {4.26, 4.136, 4.516, 5.172, 5.992, 6.946, 7.925, 8.739, 9.102,
01195     8.667, 7.534, 6.13, 4.865, 3.837, 2.924, 2.126, 1.572, 1.207,
01196     0.9693, 0.8391, 0.8016, 0.8213, 0.8584, 0.4114, 0.632, 0.9986, 1.59,
01197     2.132, 2.045, 1.617, 1.567, 1.595, 1.754, 1.846},
01198     {4.068, 3.944, 4.321, 4.953, 5.759, 6.696, 7.598, 8.349, 8.636,
01199     8.134, 6.966, 5.681, 4.531, 3.592, 2.726, 1.981, 1.454, 1.112,
01200     0.8863, 0.7642, 0.7336, 0.7503, 0.7883, 0.3649, 0.5742, 0.9245, 1.482,
01201     1.979, 1.91, 1.537, 1.492, 1.497, 1.609, 1.655},
01202     {3.784, 3.702, 4.08, 4.686, 5.436, 6.274, 7.107, 7.772, 7.978,
01203     7.457, 6.38, 5.207, 4.171, 3.317, 2.572, 1.856, 1.347, 1.012,
01204     0.8031, 0.6877, 0.6459, 0.6441, 0.6934, 0.3392, 0.5146, 0.7711, 1.206,
01205     1.627, 1.668, 1.358, 1.31, 1.315, 1.411, 1.432},
01206     {3.39, 3.409, 3.832, 4.448, 5.159, 5.932, 6.66, 7.149, 7.206,
01207     6.618, 5.602, 4.608, 3.743, 3.01, 2.347, 1.689, 1.206, 0.8923,
01208     0.6966, 0.5763, 0.5136, 0.4878, 0.5216, 0.5783, 0.3499, 0.515, 0.7012,
01209     0.9131, 1.167, 1.133, 1.139, 1.212, 1.359, 1.445},
01210     {3.031, 3.122, 3.551, 4.115, 4.781, 5.496, 6.101, 6.433, 6.32,
01211     5.654, 4.707, 3.886, 3.211, 2.629, 2.053, 1.473, 1.024, 0.7318,
01212     0.5579, 0.445, 0.3748, 0.3356, 0.3272, 0.3261, 0.3502, 0.4067, 0.4482,
01213     0.5625, 0.7534, 0.8328, 0.8615, 0.9261, 1.038, 1.075},
01214     {2.556, 2.697, 3.11, 3.64, 4.251, 4.887, 5.363, 5.492, 5.176,
01215     4.453, 3.662, 3.064, 2.599, 2.164, 1.677, 1.161, 0.7816, 0.5445,
01216     0.4076, 0.3171, 0.258, 0.2227, 0.2043, 0.1946, 0.1903, 0.2423, 0.2411,
01217     0.2984, 0.3661, 0.4305, 0.4483, 0.4735, 0.5096, 0.5082},
01218     {1.982, 2.163, 2.522, 2.962, 3.444, 3.894, 4.12, 3.996, 3.538,
01219     2.915, 2.39, 2.044, 1.761, 1.418, 1.026, 0.6684, 0.4452, 0.3147,
01220     0.2354, 0.1814, 0.1474, 0.1272, 0.1136, 0.1042, 0.09334, 0.07244, 0.09453,
01221     0.1067, 0.1323, 0.1309, 0.1255, 0.1235, 0.1251, 0.1207},
01222     {1.313, 1.48, 1.706, 1.932, 2.113, 2.193, 2.081, 1.804, 1.487,
01223     1.196, 0.9808, 0.8365, 0.6791, 0.4931, 0.3304, 0.2112, 0.1439, 0.1054,
01224     0.08052, 0.06314, 0.05248, 0.04667, 0.0419, 0.03731, 0.03192, 0.02135,

```

01225 0.01682,
 01226 0.0156, 0.01767, 0.01723, 0.0161, 0.01526, 0.0148, 0.01411},
 01227 {0.242, 0.2311, 0.2162, 0.1962, 0.1752, 0.1604, 0.1387, 0.1112, 0.08183,
 01228 0.05815, 0.04045, 0.02676, 0.01677, 0.01075, 0.007653, 0.005984, 0.00512,
 01229 0.004795,
 01230 0.004786, 0.004999, 0.004952, 0.004352, 0.003443, 0.002664, 0.002223,
 01231 0.001163, 0.001542,
 01232 0.0002821, 0.0001951, 0.000206, 0.0001656, 0.0001206, 8.303e-05,
 01233 5.901e-05},
 01234 {0.0001232, 0.0001559, 0.0001539, 0.0001693, 0.0002134, 0.0002031,
 01235 0.0001037, 1.126e-05, 5.382e-06,
 01236 1.867e-06, 5.983e-07, 2.464e-07, 1.576e-07, 1.322e-07, 1.312e-07,
 01237 1.319e-07, 3.921e-07, 3.583e-06,
 01238 3.815e-05, 6.754e-05, 0.0001004, 0.0002135, 0.0004217, 0.0007681,
 01239 0.001524, 0.0004274, 0.000876,
 01240 2.698e-05, 1.328e-12, 1.445e-13, 9.798e-14, 8.583e-14, 9.786e-14,
 01241 1.774e-13}},
 01242 {{6.595, 6.532, 7.313, 8.453, 9.864, 11.47, 13.06, 14.28, 14.67,
 01243 13.68, 11.56, 9.275, 7.452, 6.201, 5.275, 4.16, 2.898, 2.003,
 01244 1.4, 1.04, 0.7754, 0.7071, 0.7598, 0.799, 0.825, 0.9217, 1.851,
 01245 1.254, 1.138, 0.8159, 0.6311, 0.5427, 0.4614, 0.3814},
 01246 {6.516, 6.556, 7.327, 8.526, 9.924, 11.42, 12.85, 13.82, 14.03,
 01247 13.05, 11.03, 8.863, 7.108, 5.878, 4.956, 3.797, 2.704, 1.92,
 01248 1.344, 0.9685, 0.7276, 0.6364, 0.6746, 0.7239, 0.786, 0.9333, 1.793,
 01249 1.344, 1.234, 0.8885, 0.7949, 0.6932, 0.5878, 0.4871},
 01250 {6.179, 6.202, 6.853, 7.807, 8.924, 10.13, 11.21, 12.01, 12.29,
 01251 11.63, 10.05, 8.152, 6.536, 5.386, 4.503, 3.473, 2.521, 1.809,
 01252 1.273, 0.9058, 0.6837, 0.5774, 0.5746, 0.6269, 0.7726, 0.9434, 1.275,
 01253 1.102, 1.148, 0.9922, 0.9195, 0.8713, 0.8162, 0.7358},
 01254 {5.401, 5.302, 5.812, 6.634, 7.64, 8.785, 9.902, 10.82, 11.3,
 01255 10.96, 9.696, 7.981, 6.412, 5.281, 4.41, 3.469, 2.606, 1.892,
 01256 1.37, 1.034, 0.8087, 0.6766, 0.6565, 0.6981, 0.7901, 0.6904, 1.01,
 01257 1.062, 1.192, 1.063, 1.016, 0.9639, 0.8911, 0.7914},
 01258 {5.101, 4.973, 5.426, 6.18, 7.138, 8.24, 9.32, 10.29, 10.9,
 01259 10.75, 9.665, 8.035, 6.469, 5.319, 4.452, 3.502, 2.649, 1.941,
 01260 1.431, 1.09, 0.869, 0.7456, 0.7339, 0.7833, 0.8079, 1.059, 1.104,
 01261 1.303, 1.515, 1.253, 1.185, 1.131, 1.076, 0.9437},
 01262 {4.936, 4.795, 5.272, 5.985, 6.878, 7.91, 8.989, 9.922, 10.53,
 01263 10.37, 9.278, 7.698, 6.176, 5.044, 4.178, 3.263, 2.472, 1.849,
 01264 1.402, 1.087, 0.8859, 0.7846, 0.8226, 0.8854, 0.9635, 1.037, 1.251,
 01265 1.527, 1.706, 1.5, 1.503, 1.644, 1.914, 2.113},
 01266 {4.796, 4.617, 5.024, 5.703, 6.591, 7.617, 8.632, 9.544, 10.07,
 01267 9.749, 8.552, 6.983, 5.55, 4.462, 3.573, 2.707, 2.021, 1.537,
 01268 1.216, 1.017, 0.9039, 0.8702, 0.9836, 1.21, 0.6125, 1.009, 1.311,
 01269 1.688, 1.862, 1.575, 1.568, 1.696, 2.001, 2.214},
 01270 {4.522, 4.356, 4.742, 5.465, 6.36, 7.357, 8.359, 9.269, 9.706,
 01271 9.237, 7.95, 6.476, 5.137, 4.086, 3.214, 2.373, 1.75, 1.33,
 01272 1.071, 0.9379, 0.8929, 0.9071, 0.9736, 1.305, 0.5218, 1.054, 1.605,
 01273 2.105, 1.976, 1.563, 1.521, 1.56, 1.765, 1.875},
 01274 {4.201, 4.084, 4.453, 5.134, 5.998, 7.007, 8.042, 8.894, 9.218,
 01275 8.665, 7.393, 5.966, 4.728, 3.77, 2.956, 2.16, 1.585, 1.199,
 01276 0.9637, 0.8579, 0.8414, 0.8686, 0.8189, 1.154, 0.4693, 0.9934, 1.568,
 01277 2.075, 1.962, 1.563, 1.524, 1.545, 1.704, 1.786},
 01278 {3.87, 3.761, 4.135, 4.74, 5.547, 6.523, 7.533, 8.287, 8.542,
 01279 7.978, 6.743, 5.463, 4.36, 3.491, 2.739, 1.993, 1.453, 1.095,
 01280 0.8767, 0.7822, 0.7664, 0.777, 0.8145, 1.109, 0.4094, 0.8854, 1.413,
 01281 1.91, 1.872, 1.47, 1.421, 1.428, 1.538, 1.583},
 01282 {3.565, 3.517, 3.908, 4.525, 5.299, 6.159, 6.982, 7.581, 7.734,
 01283 7.15, 6.028, 4.918, 3.993, 3.242, 2.541, 1.833, 1.321, 0.9862,
 01284 0.7851, 0.6877, 0.6504, 0.6409, 0.6657, 0.7916, 0.3852, 0.627, 0.8774,
 01285 1.306, 1.713, 1.397, 1.317, 1.308, 1.379, 1.377},
 01286 {3.27, 3.307, 3.718, 4.324, 5.008, 5.72, 6.391, 6.82, 6.844,
 01287 6.25, 5.256, 4.321, 3.562, 2.929, 2.309, 1.67, 1.183, 0.8581,
 01288 0.6613, 0.5437, 0.4817, 0.4549, 0.4828, 0.4971, 0.343, 0.4517, 0.5928,
 01289 0.7482, 1.114, 1.156, 1.127, 1.142, 1.266, 1.325},
 01290 {2.881, 2.972, 3.365, 3.885, 4.479, 5.095, 5.612, 5.869, 5.739,
 01291 5.109, 4.233, 3.497, 2.928, 2.45, 1.923, 1.37, 0.937, 0.6588,
 01292 0.4974, 0.3913, 0.3216, 0.2799, 0.263, 0.2476, 0.2702, 0.3664, 0.3897,
 01293 0.4754, 0.6181, 0.6968, 0.7144, 0.7507, 0.8199, 0.8256},
 01294 {2.352, 2.522, 2.914, 3.377, 3.888, 4.391, 4.73, 4.773, 4.456,
 01295 3.814, 3.103, 2.576, 2.19, 1.824, 1.372, 0.9129, 0.606, 0.4281,
 01296 0.3241, 0.25, 0.1992, 0.1685, 0.1489, 0.1316, 0.116, 0.1598, 0.1448,
 01297 0.1805, 0.2224, 0.2379, 0.2369, 0.2454, 0.2679, 0.2718},
 01298 {1.666, 1.833, 2.135, 2.486, 2.847, 3.14, 3.202, 3.006, 2.612,
 01299 2.127, 1.726, 1.486, 1.277, 0.9733, 0.6654, 0.4233, 0.2852, 0.2051,
 01300 0.1537, 0.1174, 0.09422, 0.08017, 0.06975, 0.06009, 0.04775, 0.03319,
 01301 0.03371,
 01302 0.03896, 0.04544, 0.04203, 0.03927, 0.03814, 0.03917, 0.04012},
 01303 {0.8975, 0.9719, 1.03, 1.066, 1.034, 0.9374, 0.7957, 0.662, 0.5656,
 01304 0.4856, 0.4141, 0.3239, 0.2283, 0.1478, 0.09439, 0.06056, 0.04188,
 01305 0.03179,
 01306 0.02625, 0.02293, 0.02134, 0.01999, 0.01796, 0.01508, 0.01136, 0.006243,
 01307 0.005399,
 01308 0.002554, 0.002671, 0.002877, 0.002693, 0.002456, 0.002169, 0.001592},
 01309 {0.005568, 0.003081, 0.001936, 0.001388, 0.001138, 0.0009141, 0.0003913,
 01310 7.042e-05, 1.305e-05,
 01311 9.014e-06, 5.819e-06, 3.047e-06, 1.303e-06, 5.602e-07, 2.183e-07,

```

01312     1.757e-07, 3.825e-07, 2.566e-06,
01313     1.334e-05, 1.436e-05, 1.976e-05, 7.261e-05, 0.0002657, 0.0005962,
01314     0.001653, 0.0002773, 0.0008521,
01315     1.309e-06, 3.72e-14, 2.315e-16, 2.404e-15, 7.283e-17, 5.816e-17,
01316     1.165e-16},
01317     {5.606e-05, 7.174e-05, 7.065e-05, 8.779e-05, 0.0001175, 0.0001418,
01318     6.181e-05, 7.462e-06, 8.135e-06,
01319     6.922e-06, 3.21e-06, 1.063e-06, 3.185e-07, 7.307e-08, 1.298e-08,
01320     1.751e-08, 6.792e-08, 5.277e-07,
01321     7.612e-06, 1.832e-05, 4.78e-05, 0.0001019, 0.0001703, 0.0003801, 0.001213,
01322     0.0002105, 0.0006011,
01323     2.875e-06, 7.798e-13, 1.214e-13, 8.329e-14, 7.553e-14, 1.014e-13,
01324     1.901e-13}}
01325 };
01326
01327 #ifdef _OPENACC
01328 #pragma acc declare copyin(clim_oh_var)
01329 #endif
01330
01331 double clim_oh(
01332     double t,
01333     double lat,
01334     double p) {
01335
01336     /* Get seconds since begin of year... */
01337     double sec = FMOD(t, 365.25 * 86400.);
01338     while (sec < 0)
01339         sec += 365.25 * 86400.;
01340
01341     /* Check pressure... */
01342     if (p < clim_oh_ps[0])
01343         p = clim_oh_ps[0];
01344     else if (p > clim_oh_ps[33])
01345         p = clim_oh_ps[33];
01346
01347     /* Get indices... */
01348     int isec = locate_irr(clim_oh_secs, 12, sec);
01349     int ilat = locate_reg(clim_oh_lats, 18, lat);
01350     int ip = locate_irr(clim_oh_ps, 34, p);
01351
01352     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01353     double aux00 = LIN(clim_oh_ps[ip],
01354         clim_oh_var[isec][ilat][ip],
01355         clim_oh_ps[ip + 1],
01356         clim_oh_var[isec][ilat][ip + 1], p);
01357     double aux01 = LIN(clim_oh_ps[ip],
01358         clim_oh_var[isec][ilat + 1][ip],
01359         clim_oh_ps[ip + 1],
01360         clim_oh_var[isec][ilat + 1][ip + 1], p);
01361     double aux10 = LIN(clim_oh_ps[ip],
01362         clim_oh_var[isec + 1][ilat][ip],
01363         clim_oh_ps[ip + 1],
01364         clim_oh_var[isec + 1][ilat][ip + 1], p);
01365     double aux11 = LIN(clim_oh_ps[ip],
01366         clim_oh_var[isec + 1][ilat + 1][ip],
01367         clim_oh_ps[ip + 1],
01368         clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01369     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01370     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01371     return 1e6 * LIN(clim_oh_secs[isec], aux00,
01372         clim_oh_secs[isec + 1], aux11, sec);
01373 }
01374
01375 /*****
01376
01377 static double clim_tropo_secs[12] = {
01378     1209600.00, 3888000.00, 6393600.00,
01379     9072000.00, 11664000.00, 14342400.00,
01380     16934400.00, 19612800.00, 22291200.00,
01381     24883200.00, 27561600.00, 30153600.00
01382 };
01383
01384 #ifdef _OPENACC
01385 #pragma acc declare copyin(clim_tropo_secs)
01386 #endif
01387
01388 static double clim_tropo_lats[73]
01389 = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01390     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01391     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01392     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01393     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01394     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01395     75, 77.5, 80, 82.5, 85, 87.5, 90
01396 };
01397
01398 #ifdef _OPENACC

```

```

01399 #pragma acc declare copyin(clim_tropo_lats)
01400 #endif
01401
01402 static double clim_tropo_tps[12][73]
01403 = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01404      297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01405      175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01406      99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01407      98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01408      152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01409      277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01410      275.3, 275.6, 275.4, 274.1, 273.5},
01411 {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01412      300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01413      150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01414      98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01415      98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01416      220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01417      284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01418      287.5, 286.2, 285.8},
01419 {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01420      297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01421      161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01422      100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01423      99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01424      186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01425      279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01426      304.3, 304.9, 306, 306.6, 306.2, 306},
01427 {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01428      290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01429      195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01430      102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01431      99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01432      148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01433      263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01434      315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01435 {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01436      260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01437      205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01438      101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01439      102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01440      165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01441      273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01442      325.3, 325.8, 325.8},
01443 {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01444      222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01445      228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01446      105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01447      106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01448      127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01449      251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01450      308.5, 312.2, 313.1, 313.3},
01451 {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01452      187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01453      235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01454      110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01455      111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01456      117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01457      224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01458      275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01459 {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01460      185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01461      233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01462      110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01463      112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01464      120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01465      230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01466      278.2, 282.6, 287.4, 290.9, 292.5, 293},
01467 {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01468      183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01469      243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01470      114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01471      110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01472      114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01473      203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01474      276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01475 {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01476      215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01477      237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01478      111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01479      106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01480      112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01481      206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01482      279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01483      305.1},
01484 {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01485      253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,

```



```

01486 223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01487 108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01488 102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01489 109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01490 241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01491 286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01492 {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01493 284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01494 175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01495 100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01496 100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01497 186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01498 280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01499 281.7, 281.1, 281.2}
01500 };
01501
01502 #ifdef _OPENACC
01503 #pragma acc declare copyin(clim_tropo_tps)
01504 #endif
01505
01506 double clim_tropo(
01507     double t,
01508     double lat) {
01509
01510     /* Get seconds since begin of year... */
01511     double sec = FMOD(t, 365.25 * 86400.);
01512     while (sec < 0)
01513         sec += 365.25 * 86400.;
01514
01515     /* Get indices... */
01516     int isec = locate_irr(clim_tropo_secs, 12, sec);
01517     int ilat = locate_reg(clim_tropo_lats, 73, lat);
01518
01519     /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01520     double p0 = LIN(clim_tropo_lats[ilat],
01521                     clim_tropo_tps[isec][ilat],
01522                     clim_tropo_lats[ilat + 1],
01523                     clim_tropo_tps[isec][ilat + 1], lat);
01524     double p1 = LIN(clim_tropo_lats[ilat],
01525                     clim_tropo_tps[isec + 1][ilat],
01526                     clim_tropo_lats[ilat + 1],
01527                     clim_tropo_tps[isec + 1][ilat + 1], lat);
01528     return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01529 }
01530
01531 /*****
01532
01533 void day2doy(
01534     int year,
01535     int mon,
01536     int day,
01537     int *doy) {
01538
01539     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01540     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01541
01542     /* Get day of year... */
01543     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01544         *doy = d0l[mon - 1] + day - 1;
01545     else
01546         *doy = d0[mon - 1] + day - 1;
01547 }
01548
01549 /*****
01550
01551 void doy2day(
01552     int year,
01553     int doy,
01554     int *mon,
01555     int *day) {
01556
01557     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01558     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01559     int i;
01560
01561     /* Get month and day... */
01562     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01563         for (i = 11; i >= 0; i--)
01564             if (d0l[i] <= doy)
01565                 break;
01566         *mon = i + 1;
01567         *day = doy - d0l[i] + 1;
01568     } else {
01569         for (i = 11; i >= 0; i--)
01570             if (d0[i] <= doy)
01571                 break;
01572         *mon = i + 1;

```

```

01573     *day = doy - d0[i] + 1;
01574 }
01575 }
01576
01577 /*****
01578
01579 void geo2cart(
01580     double z,
01581     double lon,
01582     double lat,
01583     double *x) {
01584
01585     double radius = z + RE;
01586     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01587     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01588     x[2] = radius * sin(lat / 180. * M_PI);
01589 }
01590
01591 /*****
01592
01593 void get_met(
01594     ctl_t * ctl,
01595     char *metbase,
01596     double t,
01597     met_t ** met0,
01598     met_t ** met1) {
01599
01600     static int init, ip, ix, iy;
01601
01602     met_t *mets;
01603
01604     char filename[LEN];
01605
01606     /* Init... */
01607     if (t == ctl->t_start || !init) {
01608         init = 1;
01609
01610         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01611         if (!read_met(ctl, filename, *met0))
01612             ERRMSG("Cannot open file!");
01613
01614         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
dt_met, filename);
01615         if (!read_met(ctl, filename, *met1))
01616             ERRMSG("Cannot open file!");
01617 #ifdef _OPENACC
01618         met_t *met0up = *met0;
01619         met_t *met1up = *met1;
01620 #pragma acc update device(met0up[:1],met1up[:1])
01621 #endif
01622     }
01623
01624     /* Read new data for forward trajectories... */
01625     if (t > (*met1)->time && ctl->direction == 1) {
01626         mets = *met1;
01627         *met1 = *met0;
01628         *met0 = mets;
01629         get_met_help(t, 1, metbase, ctl->dt_met, filename);
01630         if (!read_met(ctl, filename, *met1))
01631             ERRMSG("Cannot open file!");
01632 #ifdef _OPENACC
01633         met_t *met1up = *met1;
01634 #pragma acc update device(met1up[:1])
01635 #endif
01636     }
01637
01638     /* Read new data for backward trajectories... */
01639     if (t < (*met0)->time && ctl->direction == -1) {
01640         mets = *met1;
01641         *met1 = *met0;
01642         *met0 = mets;
01643         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01644         if (!read_met(ctl, filename, *met0))
01645             ERRMSG("Cannot open file!");
01646 #ifdef _OPENACC
01647         met_t *met0up = *met0;
01648 #pragma acc update device(met0up[:1])
01649 #endif
01650     }
01651
01652     /* Check that grids are consistent... */
01653     if ((*met0)->nx != (*met1)->nx
01654         || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01655         ERRMSG("Meteo grid dimensions do not match!");
01656     for (ix = 0; ix < (*met0)->nx; ix++)
01657         if ((*met0)->lon[ix] != (*met1)->lon[ix])
01658             ERRMSG("Meteo grid longitudes do not match!");

```

```

01659     for (iy = 0; iy < (*met0)->ny; iy++)
01660         if ((*met0)->lat[iy] != (*met1)->lat[iy])
01661             ERRMSG("Meteo grid latitudes do not match!");
01662     for (ip = 0; ip < (*met0)->np; ip++)
01663         if ((*met0)->p[ip] != (*met1)->p[ip])
01664             ERRMSG("Meteo grid pressure levels do not match!");
01665 }
01666
01667 /*****
01668
01669 void get_met_help(
01670     double t,
01671     int direct,
01672     char *metbase,
01673     double dt_met,
01674     char *filename) {
01675     char repl[LEN];
01676     double t6, r;
01677
01678     int year, mon, day, hour, min, sec;
01679
01680     /* Round time to fixed intervals... */
01681     if (direct == -1)
01682         t6 = floor(t / dt_met) * dt_met;
01683     else
01684         t6 = ceil(t / dt_met) * dt_met;
01685
01686     /* Decode time... */
01687     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01688
01689     /* Set filename... */
01690     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01691     sprintf(repl, "%d", year);
01692     get_met_replace(filename, "YYYY", repl);
01693     sprintf(repl, "%02d", mon);
01694     get_met_replace(filename, "MM", repl);
01695     sprintf(repl, "%02d", day);
01696     get_met_replace(filename, "DD", repl);
01697     sprintf(repl, "%02d", hour);
01698     get_met_replace(filename, "HH", repl);
01699 }
01700
01701 /*****
01702
01703 void get_met_replace(
01704     char *orig,
01705     char *search,
01706     char *repl) {
01707     char buffer[LEN], *ch;
01708
01709     /* Iterate... */
01710     for (int i = 0; i < 3; i++) {
01711         /* Replace substring... */
01712         if (!(ch = strstr(orig, search)))
01713             return;
01714         strncpy(buffer, orig, (size_t) (ch - orig));
01715         buffer[ch - orig] = 0;
01716         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01717         orig[0] = 0;
01718         strcpy(orig, buffer);
01719     }
01720 }
01721
01722 /*****
01723
01724 void intpol_met_space_3d(
01725     met_t * met,
01726     float array[EX][EY][EP],
01727     double p,
01728     double lon,
01729     double lat,
01730     double *var,
01731     int *ci,
01732     double *cw,
01733     int init) {
01734     /* Check longitude... */
01735     if (met->lon[met->nx - 1] > 180 && lon < 0)
01736         lon += 360;
01737
01738     /* Get interpolation indices and weights... */
01739     if (init) {
01740         ci[0] = locate_irr(met->p, met->np, p);
01741     }
01742 }

```

```

01746     ci[1] = locate_reg(met->lon, met->nx, lon);
01747     ci[2] = locate_reg(met->lat, met->ny, lat);
01748     cw[0] = (met->p[ci[0] + 1] - p)
01749     / (met->p[ci[0] + 1] - met->p[ci[0]]);
01750     cw[1] = (met->lon[ci[1] + 1] - lon)
01751     / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01752     cw[2] = (met->lat[ci[2] + 1] - lat)
01753     / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01754 }
01755
01756 /* Interpolate vertically... */
01757 double aux00 =
01758     cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01759     + array[ci[1]][ci[2]][ci[0] + 1];
01760 double aux01 =
01761     cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01762     array[ci[1]][ci[2] + 1][ci[0] + 1])
01763     + array[ci[1]][ci[2] + 1][ci[0] + 1];
01764 double aux10 =
01765     cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01766     array[ci[1] + 1][ci[2]][ci[0] + 1])
01767     + array[ci[1] + 1][ci[2]][ci[0] + 1];
01768 double aux11 =
01769     cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01770     array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01771     + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01772
01773 /* Interpolate horizontally... */
01774 aux00 = cw[2] * (aux00 - aux01) + aux01;
01775 aux11 = cw[2] * (aux10 - aux11) + aux11;
01776 *var = cw[1] * (aux00 - aux11) + aux11;
01777 }
01778
01779
01780 /*****
01781
01782 void intpol_met_space_2d(
01783     met_t * met,
01784     float array[EX][EY],
01785     double lon,
01786     double lat,
01787     double *var,
01788     int *ci,
01789     double *cw,
01790     int init) {
01791
01792     /* Check longitude... */
01793     if (met->lon[met->nx - 1] > 180 && lon < 0)
01794         lon += 360;
01795
01796     /* Get interpolation indices and weights... */
01797     if (init) {
01798         ci[1] = locate_reg(met->lon, met->nx, lon);
01799         ci[2] = locate_reg(met->lat, met->ny, lat);
01800         cw[1] = (met->lon[ci[1] + 1] - lon)
01801         / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01802         cw[2] = (met->lat[ci[2] + 1] - lat)
01803         / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01804     }
01805
01806     /* Set variables... */
01807     double aux00 = array[ci[1]][ci[2]];
01808     double aux01 = array[ci[1]][ci[2] + 1];
01809     double aux10 = array[ci[1] + 1][ci[2]];
01810     double aux11 = array[ci[1] + 1][ci[2] + 1];
01811
01812     /* Interpolate horizontally... */
01813     if (isfinite(aux00) && isfinite(aux01))
01814         aux00 = cw[2] * (aux00 - aux01) + aux01;
01815     else if (cw[2] < 0.5)
01816         aux00 = aux01;
01817     if (isfinite(aux10) && isfinite(aux11))
01818         aux11 = cw[2] * (aux10 - aux11) + aux11;
01819     else if (cw[2] > 0.5)
01820         aux11 = aux10;
01821     if (isfinite(aux00) && isfinite(aux11))
01822         *var = cw[1] * (aux00 - aux11) + aux11;
01823     else {
01824         if (cw[1] > 0.5)
01825             *var = aux00;
01826         else
01827             *var = aux11;
01828     }
01829 }
01830
01831 /*****
01832

```

```

01833 void intpol_met_time_3d(
01834     met_t * met0,
01835     float array0[EX][EY][EP],
01836     met_t * met1,
01837     float array1[EX][EY][EP],
01838     double ts,
01839     double p,
01840     double lon,
01841     double lat,
01842     double *var,
01843     int *ci,
01844     double *cw,
01845     int init) {
01846
01847     double var0, var1, wt;
01848
01849     /* Spatial interpolation... */
01850     intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01851     intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
01852
01853     /* Get weighting factor... */
01854     wt = (met1->time - ts) / (met1->time - met0->time);
01855
01856     /* Interpolate... */
01857     *var = wt * (var0 - var1) + var1;
01858 }
01859
01860 /*****
01861 void intpol_met_time_2d(
01862     met_t * met0,
01863     float array0[EX][EY],
01864     met_t * met1,
01865     float array1[EX][EY],
01866     double ts,
01867     double lon,
01868     double lat,
01869     double *var,
01870     int *ci,
01871     double *cw,
01872     int init) {
01873
01874     double var0, var1, wt;
01875
01876     /* Spatial interpolation... */
01877     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01878     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01879
01880     /* Get weighting factor... */
01881     wt = (met1->time - ts) / (met1->time - met0->time);
01882
01883     /* Interpolate... */
01884     *var = wt * (var0 - var1) + var1;
01885 }
01886
01887 /*****
01888 void jsec2time(
01889     double jsec,
01890     int *year,
01891     int *mon,
01892     int *day,
01893     int *hour,
01894     int *min,
01895     int *sec,
01896     double *remain) {
01897
01898     struct tm t0, *t1;
01899
01900     t0.tm_year = 100;
01901     t0.tm_mon = 0;
01902     t0.tm_mday = 1;
01903     t0.tm_hour = 0;
01904     t0.tm_min = 0;
01905     t0.tm_sec = 0;
01906
01907     time_t jsec0 = (time_t) jsec + timegm(&t0);
01908     t1 = gmtime(&jsec0);
01909
01910     *year = t1->tm_year + 1900;
01911     *mon = t1->tm_mon + 1;
01912     *day = t1->tm_mday;
01913     *hour = t1->tm_hour;
01914     *min = t1->tm_min;
01915     *sec = t1->tm_sec;
01916     *remain = jsec - floor(jsec);
01917 }
01918
01919 }

```

```

01920
01921 /*****
01922
01923 int locate_irr(
01924     double **x,
01925     int n,
01926     double x) {
01927
01928     int ilo = 0;
01929     int ihi = n - 1;
01930     int i = (ihi + ilo) >> 1;
01931
01932     if (xx[i] < xx[i + 1])
01933         while (ihi > ilo + 1) {
01934             i = (ihi + ilo) >> 1;
01935             if (xx[i] > x)
01936                 ihi = i;
01937             else
01938                 ilo = i;
01939         } else
01940             while (ihi > ilo + 1) {
01941                 i = (ihi + ilo) >> 1;
01942                 if (xx[i] <= x)
01943                     ihi = i;
01944                 else
01945                     ilo = i;
01946             }
01947
01948     return ilo;
01949 }
01950
01951 /*****
01952
01953 int locate_reg(
01954     double **x,
01955     int n,
01956     double x) {
01957
01958     /* Calculate index... */
01959     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01960
01961     /* Check range... */
01962     if (i < 0)
01963         i = 0;
01964     else if (i >= n - 2)
01965         i = n - 2;
01966
01967     return i;
01968 }
01969
01970 /*****
01971
01972 int read_atm(
01973     const char *filename,
01974     ctl_t * ctl,
01975     atm_t * atm) {
01976
01977     FILE *in;
01978
01979     char line[LEN], *tok;
01980
01981     double t0;
01982
01983     int dimid, ip, iq, ncid, varid;
01984
01985     size_t nparts;
01986
01987     /* Init... */
01988     atm->np = 0;
01989
01990     /* Write info... */
01991     printf("Read atmospheric data: %s\n", filename);
01992
01993     /* Read ASCII data... */
01994     if (ctl->atm_type == 0) {
01995
01996         /* Open file... */
01997         if (!(in = fopen(filename, "r"))) {
01998             WARN("File not found!");
01999             return 0;
02000         }
02001
02002         /* Read line... */
02003         while (fgets(line, LEN, in)) {
02004
02005             /* Read data... */
02006             TOK(line, tok, "%lg", atm->time[atm->np]);

```

```

02007     TOK(NULL, tok, "%lg", atm->p[atm->np]);
02008     TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02009     TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02010     for (iq = 0; iq < ctl->nq; iq++)
02011         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02012
02013     /* Convert altitude to pressure... */
02014     atm->p[atm->np] = P(atm->p[atm->np]);
02015
02016     /* Increment data point counter... */
02017     if ((++atm->np) > NP)
02018         ERRMSG("Too many data points!");
02019 }
02020
02021 /* Close file... */
02022 fclose(in);
02023 }
02024
02025 /* Read binary data... */
02026 else if (ctl->atm_type == 1) {
02027
02028     /* Open file... */
02029     if (!(in = fopen(filename, "r")))
02030         return 0;
02031
02032     /* Read data... */
02033     FREAD(&atm->np, int, 1, in);
02034     FREAD(atm->time, double,
02035           (size_t) atm->np,
02036           in);
02037     FREAD(atm->p, double,
02038           (size_t) atm->np,
02039           in);
02040     FREAD(atm->lon, double,
02041           (size_t) atm->np,
02042           in);
02043     FREAD(atm->lat, double,
02044           (size_t) atm->np,
02045           in);
02046     for (iq = 0; iq < ctl->nq; iq++)
02047         FREAD(atm->q[iq], double,
02048               (size_t) atm->np,
02049               in);
02050
02051     /* Close file... */
02052     fclose(in);
02053 }
02054
02055 /* Read netCDF data... */
02056 else if (ctl->atm_type == 2) {
02057
02058     /* Open file... */
02059     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02060         return 0;
02061
02062     /* Get dimensions... */
02063     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02064     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02065     atm->np = (int) nparts;
02066     if (atm->np > NP)
02067         ERRMSG("Too many particles!");
02068
02069     /* Get time... */
02070     NC(nc_inq_varid(ncid, "time", &varid));
02071     NC(nc_get_var_double(ncid, varid, &t0));
02072     for (ip = 0; ip < atm->np; ip++)
02073         atm->time[ip] = t0;
02074
02075     /* Read geolocations... */
02076     NC(nc_inq_varid(ncid, "PRESS", &varid));
02077     NC(nc_get_var_double(ncid, varid, atm->p));
02078     NC(nc_inq_varid(ncid, "LON", &varid));
02079     NC(nc_get_var_double(ncid, varid, atm->lon));
02080     NC(nc_inq_varid(ncid, "LAT", &varid));
02081     NC(nc_get_var_double(ncid, varid, atm->lat));
02082
02083     /* Read variables... */
02084     if (ctl->qnt_p >= 0)
02085         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02086             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02087     if (ctl->qnt_t >= 0)
02088         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02089             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02090     if (ctl->qnt_u >= 0)
02091         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02092             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02093     if (ctl->qnt_v >= 0)

```

```

02094     if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02095         NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02096     if (ctl->qnt_w >= 0)
02097         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02098             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02099     if (ctl->qnt_h2o >= 0)
02100         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02101             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02102     if (ctl->qnt_o3 >= 0)
02103         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02104             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02105     if (ctl->qnt_theta >= 0)
02106         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02107             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02108     if (ctl->qnt_pv >= 0)
02109         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02110             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02111
02112     /* Check data... */
02113     for (ip = 0; ip < atm->np; ip++)
02114         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02115             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02116             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02117             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02118             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02119         atm->time[ip] = GSL_NAN;
02120         atm->p[ip] = GSL_NAN;
02121         atm->lon[ip] = GSL_NAN;
02122         atm->lat[ip] = GSL_NAN;
02123         for (iq = 0; iq < ctl->nq; iq++)
02124             atm->q[iq][ip] = GSL_NAN;
02125     } else {
02126         if (ctl->qnt_h2o >= 0)
02127             atm->q[ctl->qnt_h2o][ip] *= 1.608;
02128         if (ctl->qnt_pv >= 0)
02129             atm->q[ctl->qnt_pv][ip] *= 1e6;
02130         if (atm->lon[ip] > 180)
02131             atm->lon[ip] -= 360;
02132     }
02133
02134     /* Close file... */
02135     NC(nc_close(ncid));
02136 }
02137
02138 /* Error... */
02139 else
02140     ERRMSG("Atmospheric data type not supported!");
02141
02142 /* Check number of points... */
02143 if (atm->np < 1)
02144     ERRMSG("Can not read any data!");
02145
02146 /* Return success... */
02147 return 1;
02148 }
02149
02150 /*****
02151 void read_ctl(
02152     const char *filename,
02153     int argc,
02154     char *argv[],
02155     ctl_t *ctl) {
02156
02157     /* Write info... */
02158     printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02159           "(executable: %s | compiled: %s, %s)\n\n",
02160           argv[0], __DATE__, __TIME__);
02161
02162     /* Initialize quantity indices... */
02163     ctl->qnt_ens = -1;
02164     ctl->qnt_m = -1;
02165     ctl->qnt_r = -1;
02166     ctl->qnt_rho = -1;
02167     ctl->qnt_ps = -1;
02168     ctl->qnt_pt = -1;
02169     ctl->qnt_z = -1;
02170     ctl->qnt_p = -1;
02171     ctl->qnt_t = -1;
02172     ctl->qnt_u = -1;
02173     ctl->qnt_v = -1;
02174     ctl->qnt_w = -1;
02175     ctl->qnt_h2o = -1;
02176     ctl->qnt_o3 = -1;
02177     ctl->qnt_lwc = -1;
02178     ctl->qnt_iwc = -1;
02179     ctl->qnt_pc = -1;

```



```

02181     ctl->qnt_hno3 = -1;
02182     ctl->qnt_oh = -1;
02183     ctl->qnt_rh = -1;
02184     ctl->qnt_theta = -1;
02185     ctl->qnt_vh = -1;
02186     ctl->qnt_vz = -1;
02187     ctl->qnt_pv = -1;
02188     ctl->qnt_tice = -1;
02189     ctl->qnt_tsts = -1;
02190     ctl->qnt_tnat = -1;
02191     ctl->qnt_stat = -1;
02192
02193     /* Read quantities... */
02194     ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02195     if (ctl->nq > NQ)
02196         ERRMSG("Too many quantities!");
02197     for (int iq = 0; iq < ctl->nq; iq++) {
02198
02199         /* Read quantity name and format... */
02200         scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02201         scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02202             ctl->qnt_format[iq]);
02203
02204         /* Try to identify quantity... */
02205         if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
02206             ctl->qnt_ens = iq;
02207             sprintf(ctl->qnt_unit[iq], "-");
02208         } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
02209             ctl->qnt_m = iq;
02210             sprintf(ctl->qnt_unit[iq], "kg");
02211         } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
02212             ctl->qnt_r = iq;
02213             sprintf(ctl->qnt_unit[iq], "m");
02214         } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
02215             ctl->qnt_rho = iq;
02216             sprintf(ctl->qnt_unit[iq], "kg/m^3");
02217         } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
02218             ctl->qnt_ps = iq;
02219             sprintf(ctl->qnt_unit[iq], "hPa");
02220         } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
02221             ctl->qnt_pt = iq;
02222             sprintf(ctl->qnt_unit[iq], "hPa");
02223         } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
02224             ctl->qnt_z = iq;
02225             sprintf(ctl->qnt_unit[iq], "km");
02226         } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
02227             ctl->qnt_p = iq;
02228             sprintf(ctl->qnt_unit[iq], "hPa");
02229         } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
02230             ctl->qnt_t = iq;
02231             sprintf(ctl->qnt_unit[iq], "K");
02232         } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
02233             ctl->qnt_u = iq;
02234             sprintf(ctl->qnt_unit[iq], "m/s");
02235         } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
02236             ctl->qnt_v = iq;
02237             sprintf(ctl->qnt_unit[iq], "m/s");
02238         } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
02239             ctl->qnt_w = iq;
02240             sprintf(ctl->qnt_unit[iq], "hPa/s");
02241         } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
02242             ctl->qnt_h2o = iq;
02243             sprintf(ctl->qnt_unit[iq], "ppv");
02244         } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
02245             ctl->qnt_o3 = iq;
02246             sprintf(ctl->qnt_unit[iq], "ppv");
02247         } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
02248             ctl->qnt_lwc = iq;
02249             sprintf(ctl->qnt_unit[iq], "kg/kg");
02250         } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
02251             ctl->qnt_iwc = iq;
02252             sprintf(ctl->qnt_unit[iq], "kg/kg");
02253         } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
02254             ctl->qnt_pc = iq;
02255             sprintf(ctl->qnt_unit[iq], "hPa");
02256         } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
02257             ctl->qnt_hno3 = iq;
02258             sprintf(ctl->qnt_unit[iq], "ppv");
02259         } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
02260             ctl->qnt_oh = iq;
02261             sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02262         } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
02263             ctl->qnt_rh = iq;
02264             sprintf(ctl->qnt_unit[iq], "%");
02265         } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
02266             ctl->qnt_theta = iq;
02267             sprintf(ctl->qnt_unit[iq], "K");

```

```

02268     } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
02269         ctl->qnt_vh = iq;
02270         sprintf(ctl->qnt_unit[iq], "m/s");
02271     } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
02272         ctl->qnt_vz = iq;
02273         sprintf(ctl->qnt_unit[iq], "m/s");
02274     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
02275         ctl->qnt_pv = iq;
02276         sprintf(ctl->qnt_unit[iq], "PVU");
02277     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
02278         ctl->qnt_tice = iq;
02279         sprintf(ctl->qnt_unit[iq], "K");
02280     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
02281         ctl->qnt_tsts = iq;
02282         sprintf(ctl->qnt_unit[iq], "K");
02283     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
02284         ctl->qnt_tnat = iq;
02285         sprintf(ctl->qnt_unit[iq], "K");
02286     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
02287         ctl->qnt_stat = iq;
02288         sprintf(ctl->qnt_unit[iq], "-");
02289     } else
02290         scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02291 }
02292
02293 /* Time steps of simulation... */
02294 ctl->direction =
02295     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02296 if (ctl->direction != -1 && ctl->direction != 1)
02297     ERRMSG("Set DIRECTION to -1 or 1!");
02298 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02299 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
02300
02301 /* Meteorological data... */
02302 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02303 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02304 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02305 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);
02306 ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02307 ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02308 ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02309 ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02310 if (ctl->met_np > EP)
02311     ERRMSG("Too many levels!");
02312 for (int ip = 0; ip < ctl->met_np; ip++)
02313     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02314 ctl->met_tropo =
02315     (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02316 scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
02317 ctl->met_dt_out =
02318     scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02319
02320 /* Isosurface parameters... */
02321 ctl->isosurf =
02322     (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02323 scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02324
02325 /* Diffusion parameters... */
02326 ctl->turb_dx_trop =
02327     scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02328 ctl->turb_dx_strat =
02329     scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02330 ctl->turb_dz_trop =
02331     scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02332 ctl->turb_dz_strat =
02333     scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02334 ctl->turb_mesox =
02335     scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02336 ctl->turb_mesoz =
02337     scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02338
02339 /* Species parameters... */
02340 scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02341 if (strcmp(ctl->species, "SO2") == 0) {
02342     ctl->molmass = 64.066;
02343     ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02344     ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02345     ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02346     ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02347     ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
02348     ctl->wet_depo[1] = 0.62; /* (FLEXPART v10.4) */
02349     ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02350     ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02351 } else {
02352     ctl->molmass =
02353         scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02354     ctl->tdec_trop =

```

```

02355     scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02356     ctl->tdec_strat =
02357     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02358     for (int ip = 0; ip < 4; ip++)
02359         ctl->oh_chem[ip] =
02360         scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02361     for (int ip = 0; ip < 4; ip++)
02362         ctl->wet_depo[ip] =
02363         scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02364 }
02365
02366 /* PSC analysis... */
02367 ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02368 ctl->psc_hno3 =
02369     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02370
02371 /* Output of atmospheric data... */
02372 scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
02373 scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02374 ctl->atm_dt_out =
02375     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02376 ctl->atm_filter =
02377     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02378 ctl->atm_stride =
02379     (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02380 ctl->atm_type =
02381     (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02382
02383 /* Output of CSI data... */
02384 scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
02385 ctl->csi_dt_out =
02386     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02387 scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
02388 ctl->csi_obsmin =
02389     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
02390 ctl->csi_modmin =
02391     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02392 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02393 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02394 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02395 ctl->csi_lon0 =
02396     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02397 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02398 ctl->csi_nx =
02399     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02400 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02401 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02402 ctl->csi_ny =
02403     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02404
02405 /* Output of ensemble data... */
02406 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
02407
02408 /* Output of grid data... */
02409 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02410     ctl->grid_basename);
02411 scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
grid_gpfile);
02412 ctl->grid_dt_out =
02413     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02414 ctl->grid_sparse =
02415     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02416 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02417 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02418 ctl->grid_nz =
02419     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02420 ctl->grid_lon0 =
02421     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02422 ctl->grid_lon1 =
02423     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02424 ctl->grid_nx =
02425     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02426 ctl->grid_lat0 =
02427     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02428 ctl->grid_lat1 =
02429     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02430 ctl->grid_ny =
02431     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02432
02433 /* Output of profile data... */
02434 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02435     ctl->prof_basename);
02436 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->

```

```

    prof_obsfile);
02437   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02438   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02439   ctl->prof_nz =
02440       (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02441   ctl->prof_lon0 =
02442       scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02443   ctl->prof_lon1 =
02444       scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02445   ctl->prof_nx =
02446       (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02447   ctl->prof_lat0 =
02448       scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02449   ctl->prof_lat1 =
02450       scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02451   ctl->prof_ny =
02452       (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02453
02454   /* Output of station data... */
02455   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02456           ctl->stat_basename);
02457   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02458   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02459   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02460 }
02461
02462 /*****
02463
02464 int read_met(
02465     ctl_t * ctl,
02466     char *filename,
02467     met_t * met) {
02468
02469     char cmd[2 * LEN], levname[LEN], tstr[10];
02470
02471     int ip, dimid, ncid, varid, year, mon, day, hour;
02472
02473     size_t np, nx, ny;
02474
02475     /* Write info... */
02476     printf("Read meteorological data: %s\n", filename);
02477
02478     /* Get time from filename... */
02479     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
02480     year = atoi(tstr);
02481     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
02482     mon = atoi(tstr);
02483     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
02484     day = atoi(tstr);
02485     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
02486     hour = atoi(tstr);
02487     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
02488
02489     /* Open netCDF file... */
02490     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02491
02492         /* Try to stage meteo file... */
02493         if (ctl->met_stage[0] != '-') {
02494             sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
02495                     year, mon, day, hour, filename);
02496             if (system(cmd) != 0)
02497                 ERRMSG("Error while staging meteo data!");
02498         }
02499
02500         /* Try to open again... */
02501         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02502             WARN("File not found!");
02503             return 0;
02504         }
02505     }
02506
02507     /* Get dimensions... */
02508     NC(nc_inq_dimid(ncid, "lon", &dimid));
02509     NC(nc_inq_dimlen(ncid, dimid, &nx));
02510     if (nx < 2 || nx > EX)
02511         ERRMSG("Number of longitudes out of range!");
02512
02513     NC(nc_inq_dimid(ncid, "lat", &dimid));
02514     NC(nc_inq_dimlen(ncid, dimid, &ny));
02515     if (ny < 2 || ny > EY)
02516         ERRMSG("Number of latitudes out of range!");
02517
02518     sprintf(levname, "lev");
02519     NC(nc_inq_dimid(ncid, levname, &dimid));
02520     NC(nc_inq_dimlen(ncid, dimid, &np));
02521     if (np == 1) {
02522         sprintf(levname, "lev_2");

```

```

02523     if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {
02524         sprintf(levname, "plev");
02525         nc_inq_dimid(ncid, levname, &dimid);
02526     }
02527     NC(nc_inq_dimlen(ncid, dimid, &np));
02528 }
02529 if (np < 2 || np > EP)
02530     ERRMSG("Number of levels out of range!");
02531
02532 /* Store dimensions... */
02533 met->np = (int) np;
02534 met->nx = (int) nx;
02535 met->ny = (int) ny;
02536
02537 /* Get horizontal grid... */
02538 NC(nc_inq_varid(ncid, "lon", &varid));
02539 NC(nc_get_var_double(ncid, varid, met->lon));
02540 NC(nc_inq_varid(ncid, "lat", &varid));
02541 NC(nc_get_var_double(ncid, varid, met->lat));
02542
02543 /* Read meteorological data... */
02544 if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
02545     ERRMSG("Cannot read temperature!");
02546 if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
02547     ERRMSG("Cannot read zonal wind!");
02548 if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
02549     ERRMSG("Cannot read meridional wind!");
02550 if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
02551     WARN("Cannot read vertical velocity!");
02552 if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
02553     WARN("Cannot read specific humidity!");
02554 if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
02555     WARN("Cannot read ozone data!");
02556 if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
02557     WARN("Cannot read cloud liquid water content!");
02558 if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
02559     WARN("Cannot read cloud ice water content!");
02560
02561 /* Meteo data on pressure levels... */
02562 if (ctl->met_np <= 0) {
02563
02564     /* Read pressure levels from file... */
02565     NC(nc_inq_varid(ncid, levname, &varid));
02566     NC(nc_get_var_double(ncid, varid, met->p));
02567     for (ip = 0; ip < met->np; ip++)
02568         met->p[ip] /= 100.;
02569
02570     /* Extrapolate data for lower boundary... */
02571     read_met_extrapolate(met);
02572 }
02573
02574 /* Meteo data on model levels... */
02575 else {
02576
02577     /* Read pressure data from file... */
02578     read_met_help_3d(ncid, "pl", "PL", met, met->p, 0.01f);
02579
02580     /* Interpolate from model levels to pressure levels... */
02581     read_met_ml2pl(ctl, met, met->t);
02582     read_met_ml2pl(ctl, met, met->u);
02583     read_met_ml2pl(ctl, met, met->v);
02584     read_met_ml2pl(ctl, met, met->w);
02585     read_met_ml2pl(ctl, met, met->h2o);
02586     read_met_ml2pl(ctl, met, met->o3);
02587     read_met_ml2pl(ctl, met, met->lwc);
02588     read_met_ml2pl(ctl, met, met->iwc);
02589
02590     /* Set pressure levels... */
02591     met->np = ctl->met_np;
02592     for (ip = 0; ip < met->np; ip++)
02593         met->p[ip] = ctl->met_p[ip];
02594 }
02595
02596 /* Check ordering of pressure levels... */
02597 for (ip = 1; ip < met->np; ip++)
02598     if (met->p[ip - 1] < met->p[ip])
02599         ERRMSG("Pressure levels must be descending!");
02600
02601 /* Read surface data... */
02602 read_met_surface(ncid, met);
02603
02604 /* Create periodic boundary conditions... */
02605 read_met_periodic(met);
02606
02607 /* Downsampling... */
02608 read_met_sample(ctl, met);
02609

```

```

02610  /* Calculate geopotential heights... */
02611  read_met_geopot(met);
02612
02613  /* Calculate potential vorticity... */
02614  read_met_pv(met);
02615
02616  /* Calculate tropopause pressure... */
02617  read_met_tropo(ctl, met);
02618
02619  /* Calculate cloud properties... */
02620  read_met_cloud(met);
02621
02622  /* Close file... */
02623  NC(nc_close(ncid));
02624
02625  /* Return success... */
02626  return 1;
02627 }
02628
02629 /*****
02630
02631 void read_met_cloud(
02632     met_t * met) {
02633
02634     int ix, iy, ip;
02635
02636     /* Loop over columns... */
02637     #pragma omp parallel for default(shared) private(ix,iy,ip)
02638     for (ix = 0; ix < met->nx; ix++)
02639         for (iy = 0; iy < met->ny; iy++) {
02640
02641         /* Init... */
02642         met->pc[ix][iy] = GSL_NAN;
02643         met->cl[ix][iy] = 0;
02644
02645         /* Loop over pressure levels... */
02646         for (ip = 0; ip < met->np - 1; ip++) {
02647
02648             /* Check pressure... */
02649             if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02650                 continue;
02651
02652             /* Get cloud top pressure ... */
02653             if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02654                 met->pc[ix][iy] = (float) met->p[ip + 1];
02655
02656             /* Get cloud water... */
02657             met->cl[ix][iy] += (float)
02658                 (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1])
02659                  + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02660                 * 100. * (met->p[ip] - met->p[ip + 1]) / G0;
02661         }
02662     }
02663 }
02664
02665 /*****
02666
02667 void read_met_extrapolate(
02668     met_t * met) {
02669
02670     int ip, ip0, ix, iy;
02671
02672     /* Loop over columns... */
02673     #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02674     for (ix = 0; ix < met->nx; ix++)
02675         for (iy = 0; iy < met->ny; iy++) {
02676
02677             /* Find lowest valid data point... */
02678             for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02679                 if (!isfinite(met->t[ix][iy][ip0])
02680                     || !isfinite(met->u[ix][iy][ip0])
02681                     || !isfinite(met->v[ix][iy][ip0])
02682                     || !isfinite(met->w[ix][iy][ip0]))
02683                     break;
02684
02685             /* Extrapolate... */
02686             for (ip = ip0; ip >= 0; ip--) {
02687                 met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02688                 met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02689                 met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02690                 met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02691                 met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02692                 met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02693                 met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02694                 met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02695             }
02696         }

```

```

02697 }
02698
02699 /*****
02700
02701 void read_met_geopot(
02702     met_t * met) {
02703
02704     const int dx = 6, dy = 4;
02705
02706     static float help[EX][EY][EP];
02707
02708     double logp[EP], ts, z0, cw[3];
02709
02710     int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];
02711
02712     /* Calculate log pressure... */
02713     for (ip = 0; ip < met->np; ip++)
02714         logp[ip] = log(met->p[ip]);
02715
02716     /* Initialize geopotential heights... */
02717     #pragma omp parallel for default(shared) private(ix,iy,ip)
02718     for (ix = 0; ix < met->nx; ix++)
02719         for (iy = 0; iy < met->ny; iy++)
02720             for (ip = 0; ip < met->np; ip++)
02721                 met->z[ix][iy][ip] = GSL_NAN;
02722
02723     /* Apply hydrostatic equation to calculate geopotential heights... */
02724     #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip,ci,cw)
02725     for (ix = 0; ix < met->nx; ix++)
02726         for (iy = 0; iy < met->ny; iy++) {
02727
02728             /* Get surface height... */
02729             intpol_met_space_2d(met, met->z0, met->lon[ix], met->
lat[iy], &z0, ci,
02730                                 cw, 1);
02731
02732             /* Find surface pressure level index... */
02733             ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
02734
02735             /* Get virtual temperature at the surface... */
02736             ts =
02737                 LIN(met->p[ip0],
02738                     TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
02739                     met->p[ip0 + 1],
02740                     TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
02741                     met->ps[ix][iy]);
02742
02743             /* Upper part of profile... */
02744             met->z[ix][iy][ip0 + 1]
02745                 = (float) (z0 + RI / MA / G0 * 0.5
02746                     * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
02747                         met->h2o[ix][iy][ip0 + 1]))
02748                     * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
02749             for (ip = ip0 + 2; ip < met->np; ip++)
02750                 met->z[ix][iy][ip]
02751                     = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
02752                         (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
02753                             + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
02754                         * (logp[ip - 1] - logp[ip]));
02755         }
02756
02757     /* Horizontal smoothing... */
02758     #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
02759     for (ix = 0; ix < met->nx; ix++)
02760         for (iy = 0; iy < met->ny; iy++)
02761             for (ip = 0; ip < met->np; ip++) {
02762                 n = 0;
02763                 help[ix][iy][ip] = 0;
02764                 for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
02765                     ix3 = ix2;
02766                     if (ix3 < 0)
02767                         ix3 += met->nx;
02768                     else if (ix3 >= met->nx)
02769                         ix3 -= met->nx;
02770                     for (iy2 = GSL_MAX(iy - dy, 0);
02771                         iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
02772                         if (isfinite(met->z[ix3][iy2][ip])) {
02773                             help[ix][iy][ip] += met->z[ix3][iy2][ip];
02774                             n++;
02775                         }
02776                 }
02777                 if (n > 0)
02778                     help[ix][iy][ip] /= (float) n;
02779                 else
02780                     help[ix][iy][ip] = GSL_NAN;
02781             }
02782

```

```

02783  /* Copy data... */
02784 #pragma omp parallel for default(shared) private(ix,iy,ip)
02785     for (ix = 0; ix < met->nx; ix++)
02786         for (iy = 0; iy < met->ny; iy++)
02787             for (ip = 0; ip < met->np; ip++)
02788                 met->z[ix][iy][ip] = help[ix][iy][ip];
02789 }
02790
02791 /*****
02792
02793 int read_met_help_3d(
02794     int ncid,
02795     char *varname,
02796     char *varname2,
02797     met_t *met,
02798     float dest[EX][EY][EP],
02799     float scl) {
02800
02801     float *help;
02802
02803     int ip, ix, iy, varid;
02804
02805     /* Check if variable exists... */
02806     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02807         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02808             return 0;
02809
02810     /* Allocate... */
02811     ALLOC(help, float, EX * EY * EP);
02812
02813     /* Read data... */
02814     NC(nc_get_var_float(ncid, varid, help));
02815
02816     /* Copy and check data... */
02817 #pragma omp parallel for default(shared) private(ix,iy,ip)
02818     for (ix = 0; ix < met->nx; ix++)
02819         for (iy = 0; iy < met->ny; iy++)
02820             for (ip = 0; ip < met->np; ip++) {
02821                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
02822                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
02823                     dest[ix][iy][ip] *= scl;
02824                 else
02825                     dest[ix][iy][ip] = GSL_NAN;
02826             }
02827
02828     /* Free... */
02829     free(help);
02830
02831     /* Return... */
02832     return 1;
02833 }
02834
02835 /*****
02836
02837 int read_met_help_2d(
02838     int ncid,
02839     char *varname,
02840     char *varname2,
02841     met_t *met,
02842     float dest[EX][EY],
02843     float scl) {
02844
02845     float *help;
02846
02847     int ix, iy, varid;
02848
02849     /* Check if variable exists... */
02850     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02851         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02852             return 0;
02853
02854     /* Allocate... */
02855     ALLOC(help, float, EX * EY);
02856
02857     /* Read data... */
02858     NC(nc_get_var_float(ncid, varid, help));
02859
02860     /* Copy and check data... */
02861 #pragma omp parallel for default(shared) private(ix,iy)
02862     for (ix = 0; ix < met->nx; ix++)
02863         for (iy = 0; iy < met->ny; iy++) {
02864             dest[ix][iy] = help[iy * met->nx + ix];
02865             if (fabsf(dest[ix][iy]) < 1e14f)
02866                 dest[ix][iy] *= scl;
02867             else
02868                 dest[ix][iy] = GSL_NAN;
02869         }

```



```

02870
02871 /* Free... */
02872 free(help);
02873
02874 /* Return... */
02875 return 1;
02876 }
02877
02878 /*****
02879
02880 void read_met_ml2pl(
02881     ctl_t * ctl,
02882     met_t * met,
02883     float var[EX][EY][EP]) {
02884
02885     double aux[EP], p[EP], pt;
02886
02887     int ip, ip2, ix, iy;
02888
02889     /* Loop over columns... */
02890 #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
02891     for (ix = 0; ix < met->nx; ix++)
02892         for (iy = 0; iy < met->ny; iy++) {
02893
02894             /* Copy pressure profile... */
02895             for (ip = 0; ip < met->np; ip++)
02896                 p[ip] = met->p[ix][iy][ip];
02897
02898             /* Interpolate... */
02899             for (ip = 0; ip < ctl->met_np; ip++) {
02900                 pt = ctl->met_p[ip];
02901                 if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
02902                     pt = p[0];
02903                 else if ((pt > p[met->np - 1] && p[1] > p[0])
02904                     || (pt < p[met->np - 1] && p[1] < p[0]))
02905                     pt = p[met->np - 1];
02906                 ip2 = locate_irr(p, met->np, pt);
02907                 aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
02908                     p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
02909             }
02910
02911             /* Copy data... */
02912             for (ip = 0; ip < ctl->met_np; ip++)
02913                 var[ix][iy][ip] = (float) aux[ip];
02914         }
02915     }
02916
02917 /*****
02918
02919 void read_met_periodic(
02920     met_t * met) {
02921
02922     /* Check longitudes... */
02923     if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
02924         + met->lon[1] - met->lon[0] - 360) < 0.01))
02925         return;
02926
02927     /* Increase longitude counter... */
02928     if ((++met->nx) > EX)
02929         ERRMSG("Cannot create periodic boundary conditions!");
02930
02931     /* Set longitude... */
02932     met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
02933
02934     /* Loop over latitudes and pressure levels... */
02935 #pragma omp parallel for default(shared)
02936     for (int iy = 0; iy < met->ny; iy++) {
02937         met->ps[met->nx - 1][iy] = met->ps[0][iy];
02938         met->zs[met->nx - 1][iy] = met->zs[0][iy];
02939         for (int ip = 0; ip < met->np; ip++) {
02940             met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
02941             met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
02942             met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
02943             met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
02944             met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
02945             met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
02946             met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
02947             met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
02948         }
02949     }
02950 }
02951
02952 /*****
02953
02954 void read_met_pv(
02955     met_t * met) {

```

```

02956
02957 double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
02958         dtdp, dudp, dvdp, latr, vort, pows[EP];
02959
02960 int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
02961
02962 /* Set powers... */
02963 for (ip = 0; ip < met->np; ip++)
02964     pows[ip] = pow(1000. / met->p[ip], 0.286);
02965
02966 /* Loop over grid points... */
02967 #pragma omp parallel for default(shared)
02968 private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dtdx,dvdx,dtdy,dudy,dtdp,dudp,dvdp)
02969 for (ix = 0; ix < met->nx; ix++) {
02970
02971     /* Set indices... */
02972     ix0 = GSL_MAX(ix - 1, 0);
02973     ix1 = GSL_MIN(ix + 1, met->nx - 1);
02974
02975     /* Loop over grid points... */
02976     for (iy = 0; iy < met->ny; iy++) {
02977
02978         /* Set indices... */
02979         iy0 = GSL_MAX(iy - 1, 0);
02980         iy1 = GSL_MIN(iy + 1, met->ny - 1);
02981
02982         /* Set auxiliary variables... */
02983         latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
02984         dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
02985         dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
02986         c0 = cos(met->lat[iy0] / 180. * M_PI);
02987         c1 = cos(met->lat[iy1] / 180. * M_PI);
02988         cr = cos(latr / 180. * M_PI);
02989         vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
02990
02991         /* Loop over grid points... */
02992         for (ip = 0; ip < met->np; ip++) {
02993
02994             /* Get gradients in longitude... */
02995             dtdx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
02996             dvdx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
02997
02998             /* Get gradients in latitude... */
02999             dtdy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
03000             dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
03001
03002             /* Set indices... */
03003             ip0 = GSL_MAX(ip - 1, 0);
03004             ip1 = GSL_MIN(ip + 1, met->np - 1);
03005
03006             /* Get gradients in pressure... */
03007             dp0 = 100. * (met->p[ip] - met->p[ip0]);
03008             dp1 = 100. * (met->p[ip1] - met->p[ip]);
03009             if (ip != ip0 && ip != ip1) {
03010                 denom = dp0 * dp1 * (dp0 + dp1);
03011                 dtdp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03012                        - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03013                        + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03014                        / denom;
03015                 dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03016                        - dp1 * dp1 * met->u[ix][iy][ip0]
03017                        + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03018                        / denom;
03019                 dvdp = (dp0 * dp0 * met->v[ix][iy][ip1]
03020                        - dp1 * dp1 * met->v[ix][iy][ip0]
03021                        + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03022                        / denom;
03023             } else {
03024                 denom = dp0 + dp1;
03025                 dtdp =
03026                     (met->t[ix][iy][ip1] * pows[ip1] -
03027                     met->t[ix][iy][ip0] * pows[ip0]) / denom;
03028                 dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03029                 dvdp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03030             }
03031
03032             /* Calculate PV... */
03033             met->pv[ix][iy][ip] = (float)
03034                 (1e6 * G0 *
03035                  (-dtdp * (dvdx - dudy / cr + vort) + dvdp * dtdx - dudp * dtdy));
03036         }
03037     }
03038
03039     /* Fix for polar regions... */
03040 #pragma omp parallel for default(shared) private(ix,ip)
03041 for (ix = 0; ix < met->nx; ix++)

```

```

03042     for (ip = 0; ip < met->np; ip++) {
03043         met->pv[ix][0][ip]
03044         = met->pv[ix][1][ip]
03045         = met->pv[ix][2][ip];
03046         met->pv[ix][met->ny - 1][ip]
03047         = met->pv[ix][met->ny - 2][ip]
03048         = met->pv[ix][met->ny - 3][ip];
03049     }
03050 }
03051
03052 /*****
03053
03054 void read_met_sample(
03055     ctl_t * ctl,
03056     met_t * met) {
03057
03058     met_t *help;
03059
03060     float w, wsum;
03061
03062     int ip, ip2, ix, ix2, ix3, iy, iy2;
03063
03064     /* Check parameters... */
03065     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03066         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03067         return;
03068
03069     /* Allocate... */
03070     ALLOC(help, met_t, 1);
03071
03072     /* Copy data... */
03073     help->nx = met->nx;
03074     help->ny = met->ny;
03075     help->np = met->np;
03076     memcpy(help->lon, met->lon, sizeof(met->lon));
03077     memcpy(help->lat, met->lat, sizeof(met->lat));
03078     memcpy(help->p, met->p, sizeof(met->p));
03079
03080     /* Smoothing... */
03081     for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03082         for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
03083             for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03084                 help->ps[ix][iy] = 0;
03085                 help->zs[ix][iy] = 0;
03086                 help->t[ix][iy][ip] = 0;
03087                 help->u[ix][iy][ip] = 0;
03088                 help->v[ix][iy][ip] = 0;
03089                 help->w[ix][iy][ip] = 0;
03090                 help->h2o[ix][iy][ip] = 0;
03091                 help->o3[ix][iy][ip] = 0;
03092                 help->lwc[ix][iy][ip] = 0;
03093                 help->iwc[ix][iy][ip] = 0;
03094                 wsum = 0;
03095                 for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03096                     ix3 = ix2;
03097                     if (ix3 < 0)
03098                         ix3 += met->nx;
03099                     else if (ix3 >= met->nx)
03100                         ix3 -= met->nx;
03101
03102                     for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03103                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03104                         for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03105                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03106                             w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
03107                                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
03108                                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_sp);
03109                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
03110                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
03111                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03112                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03113                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03114                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03115                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03116                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03117                             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03118                             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03119                             wsum += w;
03120                         }
03121                     }
03122                 help->ps[ix][iy] /= wsum;
03123                 help->zs[ix][iy] /= wsum;
03124                 help->t[ix][iy][ip] /= wsum;
03125                 help->u[ix][iy][ip] /= wsum;
03126                 help->v[ix][iy][ip] /= wsum;
03127                 help->w[ix][iy][ip] /= wsum;
03128                 help->h2o[ix][iy][ip] /= wsum;

```

```

03129         help->o3[ix][iy][ip] /= wsum;
03130         help->lwc[ix][iy][ip] /= wsum;
03131         help->iwc[ix][iy][ip] /= wsum;
03132     }
03133 }
03134 }
03135
03136 /* Downsampling... */
03137 met->nx = 0;
03138 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03139     met->lon[met->nx] = help->lon[ix];
03140     met->ny = 0;
03141     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03142         met->lat[met->ny] = help->lat[iy];
03143         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03144         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03145         met->np = 0;
03146         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03147             met->p[met->np] = help->p[ip];
03148             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03149             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03150             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03151             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03152             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03153             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03154             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03155             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03156             met->np++;
03157         }
03158         met->ny++;
03159     }
03160     met->nx++;
03161 }
03162
03163 /* Free... */
03164 free(help);
03165 }
03166
03167 /*****
03168
03169 void read_met_surface(
03170     int ncid,
03171     met_t * met) {
03172
03173     int ix, iy;
03174
03175     /* Read surface pressure... */
03176     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03177         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03178             ERRMSG("Cannot not read surface pressure data!");
03179             for (ix = 0; ix < met->nx; ix++)
03180                 for (iy = 0; iy < met->ny; iy++)
03181                     met->ps[ix][iy] = (float) met->p[0];
03182         } else {
03183             for (iy = 0; iy < met->ny; iy++)
03184                 for (ix = 0; ix < met->nx; ix++)
03185                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03186         }
03187     }
03188
03189     /* Read geopotential height at the surface... */
03190     if (!read_met_help_2d
03191         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03192         if (!read_met_help_2d
03193             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03194             ERRMSG("Cannot read surface geopotential height!");
03195 }
03196
03197 /*****
03198
03199 void read_met_tropo(
03200     ctl_t * ctl,
03201     met_t * met) {
03202
03203     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03204         th2[200], z[EP], z2[200];
03205
03206     int found, ix, iy, iz, iz2;
03207
03208     /* Get altitude and pressure profiles... */
03209     for (iz = 0; iz < met->np; iz++)
03210         z[iz] = Z(met->p[iz]);
03211     for (iz = 0; iz <= 190; iz++) {
03212         z2[iz] = 4.5 + 0.1 * iz;
03213         p2[iz] = P(z2[iz]);
03214     }
03215

```

```

03216  /* Do not calculate tropopause... */
03217  if (ctl->met_tropo == 0)
03218      for (ix = 0; ix < met->nx; ix++)
03219          for (iy = 0; iy < met->ny; iy++)
03220              met->pt[ix][iy] = GSL_NAN;
03221
03222  /* Use tropopause climatology... */
03223  else if (ctl->met_tropo == 1) {
03224      #pragma omp parallel for default(shared) private(ix,iy)
03225          for (ix = 0; ix < met->nx; ix++)
03226              for (iy = 0; iy < met->ny; iy++)
03227                  met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03228  }
03229
03230  /* Use cold point... */
03231  else if (ctl->met_tropo == 2) {
03232
03233      /* Loop over grid points... */
03234      #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03235          for (ix = 0; ix < met->nx; ix++)
03236              for (iy = 0; iy < met->ny; iy++) {
03237
03238                  /* Interpolate temperature profile... */
03239                  for (iz = 0; iz < met->np; iz++)
03240                      t[iz] = met->t[ix][iy][iz];
03241                  spline(z, t, met->np, z2, t2, 171);
03242
03243                  /* Find minimum... */
03244                  iz = (int) gsl_stats_min_index(t2, 1, 171);
03245                  if (iz > 0 && iz < 170)
03246                      met->pt[ix][iy] = (float) p2[iz];
03247                  else
03248                      met->pt[ix][iy] = GSL_NAN;
03249              }
03250  }
03251
03252  /* Use WMO definition... */
03253  else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03254
03255      /* Loop over grid points... */
03256      #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03257          for (ix = 0; ix < met->nx; ix++)
03258              for (iy = 0; iy < met->ny; iy++) {
03259
03260                  /* Interpolate temperature profile... */
03261                  for (iz = 0; iz < met->np; iz++)
03262                      t[iz] = met->t[ix][iy][iz];
03263                  spline(z, t, met->np, z2, t2, 191);
03264
03265                  /* Find 1st tropopause... */
03266                  met->pt[ix][iy] = GSL_NAN;
03267                  for (iz = 0; iz <= 170; iz++) {
03268                      found = 1;
03269                      for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03270                          if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03271                              * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03272                              found = 0;
03273                              break;
03274                          }
03275                      if (found) {
03276                          if (iz > 0 && iz < 170)
03277                              met->pt[ix][iy] = (float) p2[iz];
03278                          break;
03279                      }
03280                  }
03281
03282                  /* Find 2nd tropopause... */
03283                  if (ctl->met_tropo == 4) {
03284                      met->pt[ix][iy] = GSL_NAN;
03285                      for (; iz <= 170; iz++) {
03286                          found = 1;
03287                          for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03288                              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03289                                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
03290                                  found = 0;
03291                                  break;
03292                              }
03293                          if (found)
03294                              break;
03295                      }
03296                      for (; iz <= 170; iz++) {
03297                          found = 1;
03298                          for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03299                              if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03300                                  * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03301                                  found = 0;
03302                                  break;

```

```

03303         }
03304         if (found) {
03305             if (iz > 0 && iz < 170)
03306                 met->pt[ix][iy] = (float) p2[iz];
03307             break;
03308         }
03309     }
03310 }
03311 }
03312 }
03313
03314 /* Use dynamical tropopause... */
03315 else if (ctl->met_tropo == 5) {
03316
03317     /* Loop over grid points... */
03318 #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03319     for (ix = 0; ix < met->nx; ix++)
03320         for (iy = 0; iy < met->ny; iy++) {
03321
03322             /* Interpolate potential vorticity profile... */
03323             for (iz = 0; iz < met->np; iz++)
03324                 pv[iz] = met->pv[ix][iy][iz];
03325             spline(z, pv, met->np, z2, pv2, 171);
03326
03327             /* Interpolate potential temperature profile... */
03328             for (iz = 0; iz < met->np; iz++)
03329                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03330             spline(z, th, met->np, z2, th2, 171);
03331
03332             /* Find dynamical tropopause 3.5 PVU + 380 K */
03333             met->pt[ix][iy] = GSL_NAN;
03334             for (iz = 0; iz <= 170; iz++)
03335                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03336                     if (iz > 0 && iz < 170)
03337                         met->pt[ix][iy] = (float) p2[iz];
03338                     break;
03339                 }
03340             }
03341         }
03342     }
03343 else
03344     ERRMSG("Cannot calculate tropopause!");
03345 }
03346
03347 /*****
03348
03349 double scan_ctl(
03350     const char *filename,
03351     int argc,
03352     char *argv[],
03353     const char *varname,
03354     int arridx,
03355     const char *defvalue,
03356     char *value) {
03357
03358     FILE *in = NULL;
03359
03360     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
03361         msg[2 * LEN], rvarname[LEN], rval[LEN];
03362
03363     int contain = 0, i;
03364
03365     /* Open file... */
03366     if (filename[strlen(filename) - 1] != '-')
03367         if (!(in = fopen(filename, "r")))
03368             ERRMSG("Cannot open file!");
03369
03370     /* Set full variable name... */
03371     if (arridx >= 0) {
03372         sprintf(fullname1, "%s[%d]", varname, arridx);
03373         sprintf(fullname2, "%s[*]", varname);
03374     } else {
03375         sprintf(fullname1, "%s", varname);
03376         sprintf(fullname2, "%s", varname);
03377     }
03378
03379     /* Read data... */
03380     if (in != NULL)
03381         while (fgets(line, LEN, in))
03382             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03383                 if (strcmp(rvarname, fullname1) == 0 ||
03384                     strcmp(rvarname, fullname2) == 0) {
03385                     contain = 1;
03386                     break;
03387                 }
03388     for (i = 1; i < argc - 1; i++)
03389         if (strcmp(argv[i], fullname1) == 0 ||

```

```

03390         strcasecmp(argv[i], fullname2) == 0) {
03391             sprintf(rval, "%s", argv[i + 1]);
03392             contain = 1;
03393             break;
03394         }
03395
03396         /* Close file... */
03397         if (in != NULL)
03398             fclose(in);
03399
03400         /* Check for missing variables... */
03401         if (!contain) {
03402             if (strlen(defvalue) > 0)
03403                 sprintf(rval, "%s", defvalue);
03404             else {
03405                 sprintf(msg, "Missing variable %s!\n", fullname1);
03406                 ERRMSG(msg);
03407             }
03408         }
03409
03410         /* Write info... */
03411         printf("%s = %s\n", fullname1, rval);
03412
03413         /* Return values... */
03414         if (value != NULL)
03415             sprintf(value, "%s", rval);
03416         return atof(rval);
03417     }
03418
03419     /*****
03420
03421 void spline(
03422     double *x,
03423     double *y,
03424     int n,
03425     double *x2,
03426     double *y2,
03427     int n2) {
03428
03429     gsl_interp_accel *acc;
03430
03431     gsl_spline *s;
03432
03433     /* Allocate... */
03434     acc = gsl_interp_accel_alloc();
03435     s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03436
03437     /* Interpolate temperature profile... */
03438     gsl_spline_init(s, x, y, (size_t) n);
03439     for (int i = 0; i < n2; i++)
03440         if (x2[i] <= x[0])
03441             y2[i] = y[0];
03442         else if (x2[i] >= x[n - 1])
03443             y2[i] = y[n - 1];
03444         else
03445             y2[i] = gsl_spline_eval(s, x2[i], acc);
03446
03447     /* Free... */
03448     gsl_spline_free(s);
03449     gsl_interp_accel_free(acc);
03450 }
03451
03452     *****/
03453
03454 double stddev(
03455     double *data,
03456     int n) {
03457
03458     if (n <= 0)
03459         return 0;
03460
03461     double avg = 0, rms = 0;
03462
03463     for (int i = 0; i < n; ++i)
03464         avg += data[i];
03465     avg /= n;
03466
03467     for (int i = 0; i < n; ++i)
03468         rms += SQR(data[i] - avg);
03469
03470     return sqrt(rms / (n - 1));
03471 }
03472
03473     *****/
03474
03475 void time2jsec(
03476     int year,

```

```

03477     int mon,
03478     int day,
03479     int hour,
03480     int min,
03481     int sec,
03482     double remain,
03483     double *jsec) {
03484
03485     struct tm t0, t1;
03486
03487     t0.tm_year = 100;
03488     t0.tm_mon = 0;
03489     t0.tm_mday = 1;
03490     t0.tm_hour = 0;
03491     t0.tm_min = 0;
03492     t0.tm_sec = 0;
03493
03494     t1.tm_year = year - 1900;
03495     t1.tm_mon = mon - 1;
03496     t1.tm_mday = day;
03497     t1.tm_hour = hour;
03498     t1.tm_min = min;
03499     t1.tm_sec = sec;
03500
03501     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03502 }
03503
03504 /*****
03505
03506 void timer(
03507     const char *name,
03508     int id,
03509     int mode) {
03510
03511     static double starttime[NTIMER], runtime[NTIMER];
03512
03513     /* Check id... */
03514     if (id < 0 || id >= NTIMER)
03515         ERRMSG("Too many timers!");
03516
03517     /* Start timer... */
03518     if (mode == 1) {
03519         if (starttime[id] <= 0)
03520             starttime[id] = omp_get_wtime();
03521         else
03522             ERRMSG("Timer already started!");
03523     }
03524
03525     /* Stop timer... */
03526     else if (mode == 2) {
03527         if (starttime[id] > 0) {
03528             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
03529             starttime[id] = -1;
03530         }
03531     }
03532
03533     /* Print timer... */
03534     else if (mode == 3) {
03535         printf("%s = %.3f s\n", name, runtime[id]);
03536         runtime[id] = 0;
03537     }
03538 }
03539
03540 /*****
03541
03542 void write_atm(
03543     const char *filename,
03544     ctl_t *ctl,
03545     atm_t *atm,
03546     double t) {
03547
03548     FILE *in, *out;
03549
03550     char line[LEN];
03551
03552     double r, t0, t1;
03553
03554     int ip, iq, year, mon, day, hour, min, sec;
03555
03556     /* Set time interval for output... */
03557     t0 = t - 0.5 * ctl->dt_mod;
03558     t1 = t + 0.5 * ctl->dt_mod;
03559
03560     /* Write info... */
03561     printf("Write atmospheric data: %s\n", filename);
03562
03563     /* Write ASCII data... */

```



```

03564     if (ctl->atm_type == 0) {
03565
03566         /* Check if gnuplot output is requested... */
03567         if (ctl->atm_gpfile[0] != '-') {
03568
03569             /* Create gnuplot pipe... */
03570             if (!(out = popen("gnuplot", "w")))
03571                 ERRMSG("Cannot create pipe to gnuplot!");
03572
03573             /* Set plot filename... */
03574             fprintf(out, "set out \"%s.png\"\n", filename);
03575
03576             /* Set time string... */
03577             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
03578             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
03579                     year, mon, day, hour, min);
03580
03581             /* Dump gnuplot file to pipe... */
03582             if (!(in = fopen(ctl->atm_gpfile, "r")))
03583                 ERRMSG("Cannot open file!");
03584             while (fgets(line, LEN, in))
03585                 fprintf(out, "%s", line);
03586             fclose(in);
03587         }
03588     }
03589     else {
03590
03591         /* Create file... */
03592         if (!(out = fopen(filename, "w")))
03593             ERRMSG("Cannot create file!");
03594     }
03595
03596     /* Write header... */
03597     fprintf(out,
03598            "# $1 = time [s]\n"
03599            "# $2 = altitude [km]\n"
03600            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03601     for (iq = 0; iq < ctl->nq; iq++)
03602         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
03603                ctl->qnt_unit[iq]);
03604     fprintf(out, "\n");
03605
03606     /* Write data... */
03607     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
03608
03609         /* Check time... */
03610         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
03611             continue;
03612
03613         /* Write output... */
03614         fprintf(out, "%.2f %g %g", atm->time[ip], Z(atm->p[ip]),
03615                atm->lon[ip], atm->lat[ip]);
03616         for (iq = 0; iq < ctl->nq; iq++) {
03617             fprintf(out, " ");
03618             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03619         }
03620         fprintf(out, "\n");
03621     }
03622
03623     /* Close file... */
03624     fclose(out);
03625 }
03626
03627 /* Write binary data... */
03628 else if (ctl->atm_type == 1) {
03629
03630     /* Create file... */
03631     if (!(out = fopen(filename, "w")))
03632         ERRMSG("Cannot create file!");
03633
03634     /* Write data... */
03635     FWRITE(&atm->np, int,
03636           1,
03637           out);
03638     FWRITE(atm->time, double,
03639           (size_t) atm->np,
03640           out);
03641     FWRITE(atm->p, double,
03642           (size_t) atm->np,
03643           out);
03644     FWRITE(atm->lon, double,
03645           (size_t) atm->np,
03646           out);
03647     FWRITE(atm->lat, double,
03648           (size_t) atm->np,
03649           out);
03650     for (iq = 0; iq < ctl->nq; iq++)

```

```

03651     FWRITE(atm->q[iq], double,
03652             (size_t) atm->np,
03653             out);
03654
03655     /* Close file... */
03656     fclose(out);
03657 }
03658
03659 /* Error... */
03660 else
03661     ERRMSG("Atmospheric data type not supported!");
03662 }
03663
03664 /*****
03665
03666 void write_csi(
03667     const char *filename,
03668     ctl_t * ctl,
03669     atm_t * atm,
03670     double t) {
03671
03672     static FILE *in, *out;
03673
03674     static char line[LEN];
03675
03676     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
03677         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
03678
03679     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
03680
03681     /* Init... */
03682     if (t == ctl->t_start) {
03683
03684         /* Check quantity index for mass... */
03685         if (ctl->qnt_m < 0)
03686             ERRMSG("Need quantity mass!");
03687
03688         /* Open observation data file... */
03689         printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
03690         if (!(in = fopen(ctl->csi_obsfile, "r")))
03691             ERRMSG("Cannot open file!");
03692
03693         /* Create new file... */
03694         printf("Write CSI data: %s\n", filename);
03695         if (!(out = fopen(filename, "w")))
03696             ERRMSG("Cannot create file!");
03697
03698         /* Write header... */
03699         fprintf(out,
03700             "# $1 = time [s]\n"
03701             "# $2 = number of hits (cx)\n"
03702             "# $3 = number of misses (cy)\n"
03703             "# $4 = number of false alarms (cz)\n"
03704             "# $5 = number of observations (cx + cy)\n"
03705             "# $6 = number of forecasts (cx + cz)\n"
03706             "# $7 = bias (forecasts/observations) [%]\n"
03707             "# $8 = probability of detection (POD) [%]\n"
03708             "# $9 = false alarm rate (FAR) [%]\n"
03709             "# $10 = critical success index (CSI) [%]\n\n");
03710     }
03711
03712     /* Set time interval... */
03713     t0 = t - 0.5 * ctl->dt_mod;
03714     t1 = t + 0.5 * ctl->dt_mod;
03715
03716     /* Initialize grid cells... */
03717     #pragma omp parallel for default(shared) private(ix,iy,iz)
03718     for (ix = 0; ix < ctl->csi_nx; ix++)
03719         for (iy = 0; iy < ctl->csi_ny; iy++)
03720             for (iz = 0; iz < ctl->csi_nz; iz++)
03721                 modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
03722
03723     /* Read observation data... */
03724     while (fgets(line, LEN, in)) {
03725
03726         /* Read data... */
03727         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
03728             5)
03729             continue;
03730
03731         /* Check time... */
03732         if (rt < t0)
03733             continue;
03734         if (rt > t1)
03735             break;
03736
03737         /* Calculate indices... */

```

```

03738     ix = (int) ((rlon - ctl->csi_lon0)
03739                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03740     iy = (int) ((rlat - ctl->csi_lat0)
03741                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03742     iz = (int) ((rz - ctl->csi_z0)
03743                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03744
03745     /* Check indices... */
03746     if (ix < 0 || ix >= ctl->csi_nx ||
03747         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03748         continue;
03749
03750     /* Get mean observation index... */
03751     obsmean[ix][iy][iz] += robs;
03752     obscount[ix][iy][iz]++;
03753 }
03754
03755 /* Analyze model data... */
03756 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03757 for (ip = 0; ip < atm->np; ip++) {
03758
03759     /* Check time... */
03760     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03761         continue;
03762
03763     /* Get indices... */
03764     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
03765                / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03766     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
03767                / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03768     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
03769                / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03770
03771     /* Check indices... */
03772     if (ix < 0 || ix >= ctl->csi_nx ||
03773         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03774         continue;
03775
03776     /* Get total mass in grid cell... */
03777     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03778 }
03779
03780 /* Analyze all grid cells... */
03781 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
03782 for (ix = 0; ix < ctl->csi_nx; ix++)
03783     for (iy = 0; iy < ctl->csi_ny; iy++)
03784         for (iz = 0; iz < ctl->csi_nz; iz++) {
03785
03786             /* Calculate mean observation index... */
03787             if (obscount[ix][iy][iz] > 0)
03788                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
03789
03790             /* Calculate column density... */
03791             if (modmean[ix][iy][iz] > 0) {
03792                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
03793                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
03794                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
03795                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
03796                     * cos(lat * M_PI / 180.);
03797                 modmean[ix][iy][iz] /= (1e6 * area);
03798             }
03799
03800             /* Calculate CSI... */
03801             if (obscount[ix][iy][iz] > 0) {
03802                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03803                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03804                     cx++;
03805                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03806                     modmean[ix][iy][iz] < ctl->csi_modmin)
03807                     cy++;
03808                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
03809                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03810                     cz++;
03811             }
03812         }
03813
03814     /* Write output... */
03815     if (fmod(t, ctl->csi_dt_out) == 0) {
03816
03817         /* Write... */
03818         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
03819                t, cx, cy, cz, cx + cy, cx + cz,
03820                (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
03821                (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
03822                (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
03823                (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
03824

```

```

03825     /* Set counters to zero... */
03826     cx = cy = cz = 0;
03827 }
03828
03829 /* Close file... */
03830 if (t == ctl->t_stop)
03831     fclose(out);
03832 }
03833
03834 /*****
03835
03836 void write_ens(
03837     const char *filename,
03838     ctl_t * ctl,
03839     atm_t * atm,
03840     double t) {
03841
03842     static FILE *out;
03843
03844     static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
03845         t0, t1, x[NENS][3], xm[3];
03846
03847     static int ip, iq;
03848
03849     static size_t i, n;
03850
03851     /* Init... */
03852     if (t == ctl->t_start) {
03853
03854         /* Check quantities... */
03855         if (ctl->qnt_ens < 0)
03856             ERRMSG("Missing ensemble IDs!");
03857
03858         /* Create new file... */
03859         printf("Write ensemble data: %s\n", filename);
03860         if (!(out = fopen(filename, "w")))
03861             ERRMSG("Cannot create file!");
03862
03863         /* Write header... */
03864         fprintf(out,
03865             "# $1 = time [s]\n"
03866             "# $2 = altitude [km]\n"
03867             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03868         for (iq = 0; iq < ctl->nq; iq++)
03869             fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
03870                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03871         for (iq = 0; iq < ctl->nq; iq++)
03872             fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
03873                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03874         fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
03875     }
03876
03877     /* Set time interval... */
03878     t0 = t - 0.5 * ctl->dt_mod;
03879     t1 = t + 0.5 * ctl->dt_mod;
03880
03881     /* Init... */
03882     ens = GSL_NAN;
03883     n = 0;
03884
03885     /* Loop over air parcels... */
03886     for (ip = 0; ip < atm->np; ip++) {
03887
03888         /* Check time... */
03889         if (atm->time[ip] < t0 || atm->time[ip] > t1)
03890             continue;
03891
03892         /* Check ensemble id... */
03893         if (atm->q[ctl->qnt_ens][ip] != ens) {
03894
03895             /* Write results... */
03896             if (n > 0) {
03897
03898                 /* Get mean position... */
03899                 xm[0] = xm[1] = xm[2] = 0;
03900                 for (i = 0; i < n; i++) {
03901                     xm[0] += x[i][0] / (double) n;
03902                     xm[1] += x[i][1] / (double) n;
03903                     xm[2] += x[i][2] / (double) n;
03904                 }
03905                 cart2geo(xm, &dummy, &lon, &lat);
03906                 fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
03907                     lat);
03908
03909                 /* Get quantity statistics... */
03910                 for (iq = 0; iq < ctl->nq; iq++) {
03911                     fprintf(out, " ");

```

```

03912         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03913     }
03914     for (iq = 0; iq < ctl->nq; iq++) {
03915         fprintf(out, " ");
03916         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03917     }
03918     fprintf(out, " %lu\n", n);
03919 }
03920
03921 /* Init new ensemble... */
03922 ens = atm->q[ctl->qnt_ens][ip];
03923 n = 0;
03924 }
03925
03926 /* Save data... */
03927 p[n] = atm->p[ip];
03928 geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
03929 for (iq = 0; iq < ctl->nq; iq++)
03930     q[iq][n] = atm->q[iq][ip];
03931 if ((++n) >= NENS)
03932     ERRMSG("Too many data points!");
03933 }
03934
03935 /* Write results... */
03936 if (n > 0) {
03937     /* Get mean position... */
03938     xm[0] = xm[1] = xm[2] = 0;
03939     for (i = 0; i < n; i++) {
03940         xm[0] += x[i][0] / (double) n;
03941         xm[1] += x[i][1] / (double) n;
03942         xm[2] += x[i][2] / (double) n;
03943     }
03944     cart2geo(xm, &dummy, &lon, &lat);
03945     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
03946
03947     /* Get quantity statistics... */
03948     for (iq = 0; iq < ctl->nq; iq++) {
03949         fprintf(out, " ");
03950         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03951     }
03952     for (iq = 0; iq < ctl->nq; iq++) {
03953         fprintf(out, " ");
03954         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03955     }
03956     fprintf(out, " %lu\n", n);
03957 }
03958
03959 /* Close file... */
03960 if (t == ctl->t_stop)
03961     fclose(out);
03962 }
03963
03964
03965 /*****
03966
03967 void write_grid(
03968     const char *filename,
03969     ctl_t * ctl,
03970     met_t * met0,
03971     met_t * met1,
03972     atm_t * atm,
03973     double t) {
03974
03975     FILE *in, *out;
03976
03977     char line[LEN];
03978
03979     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
03980         area, rho_air, press, temp, cd, vmr, t0, t1, r, cw[3];
03981
03982     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
03983         ci[3];
03984
03985     /* Check dimensions... */
03986     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
03987         ERRMSG("Grid dimensions too large!");
03988
03989     /* Set time interval for output... */
03990     t0 = t - 0.5 * ctl->dt_mod;
03991     t1 = t + 0.5 * ctl->dt_mod;
03992
03993     /* Set grid box size... */
03994     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
03995     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
03996     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
03997
03998     /* Initialize grid... */

```

```

03999 #pragma omp parallel for default(shared) private(ix,iy,iz)
04000     for (ix = 0; ix < ctl->grid_nx; ix++)
04001         for (iy = 0; iy < ctl->grid_ny; iy++)
04002             for (iz = 0; iz < ctl->grid_nz; iz++) {
04003                 mass[ix][iy][iz] = 0;
04004                 np[ix][iy][iz] = 0;
04005             }
04006
04007     /* Average data... */
04008 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04009     for (ip = 0; ip < atm->np; ip++)
04010         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04011
04012             /* Get index... */
04013             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04014             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04015             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04016
04017             /* Check indices... */
04018             if (ix < 0 || ix >= ctl->grid_nx ||
04019                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04020                 continue;
04021
04022             /* Add mass... */
04023             if (ctl->qnt_m >= 0)
04024                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04025             np[ix][iy][iz]++;
04026         }
04027
04028     /* Check if gnuplot output is requested... */
04029     if (ctl->grid_gpfile[0] != '-') {
04030
04031         /* Write info... */
04032         printf("Plot grid data: %s.png\n", filename);
04033
04034         /* Create gnuplot pipe... */
04035         if (!(out = popen("gnuplot", "w")))
04036             ERRMSG("Cannot create pipe to gnuplot!");
04037
04038         /* Set plot filename... */
04039         fprintf(out, "set out \"%s.png\"\n", filename);
04040
04041         /* Set time string... */
04042         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04043         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04044             year, mon, day, hour, min);
04045
04046         /* Dump gnuplot file to pipe... */
04047         if (!(in = fopen(ctl->grid_gpfile, "r")))
04048             ERRMSG("Cannot open file!");
04049         while (fgets(line, LEN, in))
04050             fprintf(out, "%s", line);
04051         fclose(in);
04052     }
04053
04054     else {
04055
04056         /* Write info... */
04057         printf("Write grid data: %s\n", filename);
04058
04059         /* Create file... */
04060         if (!(out = fopen(filename, "w")))
04061             ERRMSG("Cannot create file!");
04062     }
04063
04064     /* Write header... */
04065     fprintf(out,
04066         "## $1 = time [s]\n"
04067         "## $2 = altitude [km]\n"
04068         "## $3 = longitude [deg]\n"
04069         "## $4 = latitude [deg]\n"
04070         "## $5 = surface area [km^2]\n"
04071         "## $6 = layer width [km]\n"
04072         "## $7 = number of particles [l]\n"
04073         "## $8 = column density [kg/m^2]\n"
04074         "## $9 = volume mixing ratio [ppv]\n\n");
04075
04076     /* Write data... */
04077     for (ix = 0; ix < ctl->grid_nx; ix++) {
04078         if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04079             fprintf(out, "\n");
04080         for (iy = 0; iy < ctl->grid_ny; iy++) {
04081             if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
04082                 fprintf(out, "\n");
04083             for (iz = 0; iz < ctl->grid_nz; iz++)
04084                 if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04085

```

```

04086      /* Set coordinates... */
04087      z = ctl->grid_z0 + dz * (iz + 0.5);
04088      lon = ctl->grid_lon0 + dlon * (ix + 0.5);
04089      lat = ctl->grid_lat0 + dlat * (iy + 0.5);
04090
04091      /* Get pressure and temperature... */
04092      press = P(z);
04093      intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04094                        lat, &temp, ci, cw, 1);
04095
04096      /* Calculate surface area... */
04097      area = dlat * dlon * SQR(RE * M_PI / 180.)
04098            * cos(lat * M_PI / 180.);
04099
04100      /* Calculate column density... */
04101      cd = mass[ix][iy][iz] / (le6 * area);
04102
04103      /* Calculate volume mixing ratio... */
04104      rho_air = 100. * press / (RA * temp);
04105      vmr = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
04106            / (rho_air * le6 * area * le3 * dz) : GSL_NAN;
04107
04108      /* Write output... */
04109      fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
04110            t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
04111    }
04112  }
04113 }
04114
04115 /* Close file... */
04116 fclose(out);
04117 }
04118
04119 /*****
04120
04121 void write_prof(
04122   const char *filename,
04123   ctl_t * ctl,
04124   met_t * met0,
04125   met_t * met1,
04126   atm_t * atm,
04127   double t) {
04128
04129   static FILE *in, *out;
04130
04131   static char line[LEN];
04132
04133   static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
04134     t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
04135     o3, cw[3];
04136
04137   static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];
04138
04139   /* Init... */
04140   if (t == ctl->t_start) {
04141
04142     /* Check quantity index for mass... */
04143     if (ctl->qnt_m < 0)
04144       ERRMSG("Need quantity mass!");
04145
04146     /* Check dimensions... */
04147     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04148       ERRMSG("Grid dimensions too large!");
04149
04150     /* Check molar mass... */
04151     if (ctl->molmass <= 0)
04152       ERRMSG("Specify molar mass!");
04153
04154     /* Open observation data file... */
04155     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04156     if (!(in = fopen(ctl->prof_obsfile, "r")))
04157       ERRMSG("Cannot open file!");
04158
04159     /* Create new output file... */
04160     printf("Write profile data: %s\n", filename);
04161     if (!(out = fopen(filename, "w")))
04162       ERRMSG("Cannot create file!");
04163
04164     /* Write header... */
04165     fprintf(out,
04166           "# $1 = time [s]\n"
04167           "# $2 = altitude [km]\n"
04168           "# $3 = longitude [deg]\n"
04169           "# $4 = latitude [deg]\n"
04170           "# $5 = pressure [hPa]\n"
04171           "# $6 = temperature [K]\n"
04172           "# $7 = volume mixing ratio [ppv]\n"

```

```

04173         "# $8 = H2O volume mixing ratio [ppv]\n"
04174         "# $9 = O3 volume mixing ratio [ppv]\n"
04175         "# $10 = observed BT index [K]\n");
04176
04177     /* Set grid box size... */
04178     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04179     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04180     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04181 }
04182
04183     /* Set time interval... */
04184     t0 = t - 0.5 * ctl->dt_mod;
04185     t1 = t + 0.5 * ctl->dt_mod;
04186
04187     /* Initialize... */
04188 #pragma omp parallel for default(shared) private(ix,iy,iz)
04189     for (ix = 0; ix < ctl->prof_nx; ix++)
04190         for (iy = 0; iy < ctl->prof_ny; iy++) {
04191             obsmean[ix][iy] = 0;
04192             obscount[ix][iy] = 0;
04193             for (iz = 0; iz < ctl->prof_nz; iz++)
04194                 mass[ix][iy][iz] = 0;
04195         }
04196
04197     /* Read observation data... */
04198     while (fgets(line, LEN, in)) {
04199
04200         /* Read data... */
04201         if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &robs) !=
04202             5)
04203             continue;
04204
04205         /* Check time... */
04206         if (rt < t0)
04207             continue;
04208         if (rt > t1)
04209             break;
04210
04211         /* Calculate indices... */
04212         ix = (int) ((rln - ctl->prof_lon0) / dlon);
04213         iy = (int) ((rln - ctl->prof_lat0) / dlat);
04214
04215         /* Check indices... */
04216         if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04217             continue;
04218
04219         /* Get mean observation index... */
04220         obsmean[ix][iy] += robs;
04221         obscount[ix][iy]++;
04222     }
04223
04224     /* Analyze model data... */
04225 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04226     for (ip = 0; ip < atm->np; ip++) {
04227
04228         /* Check time... */
04229         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04230             continue;
04231
04232         /* Get indices... */
04233         ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04234         iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04235         iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04236
04237         /* Check indices... */
04238         if (ix < 0 || ix >= ctl->prof_nx ||
04239             iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04240             continue;
04241
04242         /* Get total mass in grid cell... */
04243         mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04244     }
04245
04246     /* Extract profiles... */
04247     for (ix = 0; ix < ctl->prof_nx; ix++)
04248         for (iy = 0; iy < ctl->prof_ny; iy++)
04249             if (obscount[ix][iy] > 0) {
04250
04251                 /* Check profile... */
04252                 okay = 0;
04253                 for (iz = 0; iz < ctl->prof_nz; iz++)
04254                     if (mass[ix][iy][iz] > 0) {
04255                         okay = 1;
04256                         break;
04257                     }
04258                 if (!okay)
04259                     continue;

```



```

04260
04261     /* Write output... */
04262     fprintf(out, "\n");
04263
04264     /* Loop over altitudes... */
04265     for (iz = 0; iz < ctl->prof_nz; iz++) {
04266
04267         /* Set coordinates... */
04268         z = ctl->prof_z0 + dz * (iz + 0.5);
04269         lon = ctl->prof_lon0 + dlon * (ix + 0.5);
04270         lat = ctl->prof_lat0 + dlat * (iy + 0.5);
04271
04272         /* Get pressure and temperature... */
04273         press = P(z);
04274         intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04275                           lat, &temp, ci, cw, 1);
04276         intpol_met_time_3d(met0, met0->h2o, met1, met1->
04277                           h2o, t, press, lon,
04278                           lat, &h2o, ci, cw, 0);
04279         intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
04280                           lat, &o3, ci, cw, 0);
04281
04282         /* Calculate surface area... */
04283         area = dlat * dlon * SQR(M_PI * RE / 180.)
04284               * cos(lat * M_PI / 180.);
04285
04286         /* Calculate volume mixing ratio... */
04287         rho_air = 100. * press / (RA * temp);
04288         vmr = MA / ctl->molmass * mass[ix][iy][iz]
04289              / (rho_air * area * dz * 1e9);
04290
04291         /* Write output... */
04292         fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
04293               t, z, lon, lat, press, temp, vmr, h2o, o3,
04294               obsmean[ix][iy] / obscount[ix][iy]);
04295     }
04296 }
04297
04298 /* Close file... */
04299 if (t == ctl->t_stop)
04300     fclose(out);
04301 }
04302
04303 /*****
04304 void write_station(
04305     const char *filename,
04306     ctl_t * ctl,
04307     atm_t * atm,
04308     double t) {
04309
04310     static FILE *out;
04311
04312     static double rmax2, t0, t1, x0[3], x1[3];
04313
04314     /* Init... */
04315     if (t == ctl->t_start) {
04316
04317         /* Write info... */
04318         printf("Write station data: %s\n", filename);
04319
04320         /* Create new file... */
04321         if (!(out = fopen(filename, "w")))
04322             ERRMSG("Cannot create file!");
04323
04324         /* Write header... */
04325         fprintf(out,
04326               "# $1 = time [s]\n"
04327               "# $2 = altitude [km]\n"
04328               "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04329         for (int iq = 0; iq < ctl->nq; iq++)
04330             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
04331                   ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04332         fprintf(out, "\n");
04333
04334         /* Set geolocation and search radius... */
04335         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
04336         rmax2 = SQR(ctl->stat_r);
04337     }
04338
04339     /* Set time interval for output... */
04340     t0 = t - 0.5 * ctl->dt_mod;
04341     t1 = t + 0.5 * ctl->dt_mod;
04342
04343     /* Loop over air parcels... */
04344     for (int ip = 0; ip < atm->np; ip++) {
04345

```

```

04346      /* Check time... */
04347      if (atm->time[ip] < t0 || atm->time[ip] > t1)
04348          continue;
04349
04350      /* Check station flag... */
04351      if (ctl->qnt_stat >= 0)
04352          if (atm->q[ctl->qnt_stat][ip])
04353              continue;
04354
04355      /* Get Cartesian coordinates... */
04356      geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04357
04358      /* Check horizontal distance... */
04359      if (DIST2(x0, x1) > rmax2)
04360          continue;
04361
04362      /* Set station flag... */
04363      if (ctl->qnt_stat >= 0)
04364          atm->q[ctl->qnt_stat][ip] = 1;
04365
04366      /* Write data... */
04367      fprintf(out, "%.2f %g %g %g",
04368              atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
04369      for (int iq = 0; iq < ctl->nq; iq++) {
04370          fprintf(out, " ");
04371          fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04372      }
04373      fprintf(out, "\n");
04374  }
04375
04376      /* Close file... */
04377      if (t == ctl->t_stop)
04378          fclose(out);
04379  }

```

5.21 libtrac.h File Reference

MPTRAC library declarations.

Data Structures

- struct [ctl_t](#)
Control parameters.
- struct [atm_t](#)
Atmospheric data.
- struct [cache_t](#)
Cache data.
- struct [met_t](#)
Meteorological data.

Functions

- void [cart2geo](#) (double *x, double *z, double *lon, double *lat)
Convert Cartesian coordinates to geolocation.
- int [check_finite](#) (const double x)
Check if x is finite.
- double [clim_hno3](#) (double t, double lat, double p)
Climatology of HNO₃ volume mixing ratios.
- double [clim_oh](#) (double t, double lat, double p)
Climatology of OH number concentrations.
- double [clim_tropo](#) (double t, double lat)
Climatology of tropopause pressure.

- void [day2doy](#) (int year, int mon, int day, int *doy)
Get day of year from date.
- void [doy2day](#) (int year, int doy, int *mon, int *day)
Get date from day of year.
- void [geo2cart](#) (double z, double lon, double lat, double *x)
Convert geolocation to Cartesian coordinates.
- void [get_met](#) (ctl_t *ctl, char *metbase, double t, met_t **met0, met_t **met1)
Get meteorological data for given timestep.
- void [get_met_help](#) (double t, int direct, char *metbase, double dt_met, char *filename)
Get meteorological data for timestep.
- void [get_met_replace](#) (char *orig, char *search, char *repl)
Replace template strings in filename.
- void [intpol_met_space_3d](#) (met_t *met, float array[EX][EY][EP], double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteorological data.
- void [intpol_met_space_2d](#) (met_t *met, float array[EX][EY], double lon, double lat, double *var, int *ci, double *cw, int init)
Spatial interpolation of meteorological data.
- void [intpol_met_time_3d](#) (met_t *met0, float array0[EX][EY][EP], met_t *met1, float array1[EX][EY][EP], double ts, double p, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void [intpol_met_time_2d](#) (met_t *met0, float array0[EX][EY], met_t *met1, float array1[EX][EY], double ts, double lon, double lat, double *var, int *ci, double *cw, int init)
Temporal interpolation of meteorological data.
- void [jsec2time](#) (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)
Convert seconds to date.
- int [locate_irr](#) (double *xx, int n, double x)
Find array index for irregular grid.
- int [locate_reg](#) (double *xx, int n, double x)
Find array index for regular grid.
- int [read_atm](#) (const char *filename, ctl_t *ctl, atm_t *atm)
Read atmospheric data.
- void [read_ctl](#) (const char *filename, int argc, char *argv[], ctl_t *ctl)
Read control parameters.
- int [read_met](#) (ctl_t *ctl, char *filename, met_t *met)
Read meteorological data file.
- void [read_met_cloud](#) (met_t *met)
Calculate cloud properties.
- void [read_met_extrapolate](#) (met_t *met)
Extrapolate meteorological data at lower boundary.
- void [read_met_geopot](#) (met_t *met)
Calculate geopotential heights.
- int [read_met_help_3d](#) (int ncid, char *varname, char *varname2, met_t *met, float dest[EX][EY][EP], float scl)
Read and convert 3D variable from meteorological data file.
- int [read_met_help_2d](#) (int ncid, char *varname, char *varname2, met_t *met, float dest[EX][EY], float scl)
Read and convert 2D variable from meteorological data file.
- void [read_met_ml2pl](#) (ctl_t *ctl, met_t *met, float var[EX][EY][EP])
Convert meteorological data from model levels to pressure levels.
- void [read_met_periodic](#) (met_t *met)
Create meteorological data with periodic boundary conditions.

- void `read_met_pv` (`met_t` *met)
Calculate potential vorticity.
- void `read_met_sample` (`ctl_t` *ctl, `met_t` *met)
Downsampling of meteorological data.
- void `read_met_surface` (int ncid, `met_t` *met)
Read surface data.
- void `read_met_tropo` (`ctl_t` *ctl, `met_t` *met)
Calculate tropopause pressure.
- double `scan_ctl` (const char *filename, int argc, char *argv[], const char *varname, int arridx, const char *defvalue, char *value)
Read a control parameter from file or command line.
- void `spline` (double *x, double *y, int n, double *x2, double *y2, int n2)
Spline interpolation.
- double `stddev` (double *data, int n)
Calculate standard deviation.
- void `time2jsec` (int year, int mon, int day, int hour, int min, int sec, double remain, double *jsec)
Convert date to seconds.
- void `timer` (const char *name, int id, int mode)
Measure wall-clock time.
- void `write_atm` (const char *filename, `ctl_t` *ctl, `atm_t` *atm, double t)
Write atmospheric data.
- void `write_csi` (const char *filename, `ctl_t` *ctl, `atm_t` *atm, double t)
Write CSI data.
- void `write_ens` (const char *filename, `ctl_t` *ctl, `atm_t` *atm, double t)
Write ensemble data.
- void `write_grid` (const char *filename, `ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double t)
Write gridded data.
- void `write_prof` (const char *filename, `ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, double t)
Write profile data.
- void `write_station` (const char *filename, `ctl_t` *ctl, `atm_t` *atm, double t)
Write station data.

5.21.1 Detailed Description

MPTRAC library declarations.

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

5.21.2 Function Documentation

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

Convert Cartesian coordinates to geolocation.

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

```

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

5.21.2.2 int check_finite (const double x)

Check if x is finite.

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

Climatology of HNO3 volume mixing ratios.

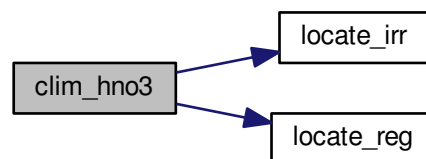
Definition at line 304 of file [libtrac.c](#).

```

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

```

Here is the call graph for this function:



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

Climatology of OH number concentrations.

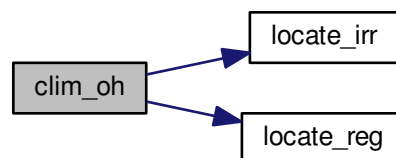
Definition at line 1331 of file libtrac.c.

```

01334     {
01335
01336     /* Get seconds since begin of year... */
01337     double sec = FMOD(t, 365.25 * 86400.);
01338     while (sec < 0)
01339         sec += 365.25 * 86400.;
01340
01341     /* Check pressure... */
01342     if (p < clim_oh_ps[0])
01343         p = clim_oh_ps[0];
01344     else if (p > clim_oh_ps[33])
01345         p = clim_oh_ps[33];
01346
01347     /* Get indices... */
01348     int isec = locate_irr(clim_oh_secs, 12, sec);
01349     int ilat = locate_reg(clim_oh_lats, 18, lat);
01350     int ip = locate_irr(clim_oh_ps, 34, p);
01351
01352     /* Interpolate OH climatology (Pommrich et al., 2014)... */
01353     double aux00 = LIN(clim_oh_ps[ip],
01354                        clim_oh_var[isec][ilat][ip],
01355                        clim_oh_ps[ip + 1],
01356                        clim_oh_var[isec][ilat][ip + 1], p);
01357     double aux01 = LIN(clim_oh_ps[ip],
01358                        clim_oh_var[isec][ilat + 1][ip],
01359                        clim_oh_ps[ip + 1],
01360                        clim_oh_var[isec][ilat + 1][ip + 1], p);
01361     double aux10 = LIN(clim_oh_ps[ip],
01362                        clim_oh_var[isec + 1][ilat][ip],
01363                        clim_oh_ps[ip + 1],
01364                        clim_oh_var[isec + 1][ilat][ip + 1], p);
01365     double aux11 = LIN(clim_oh_ps[ip],
01366                        clim_oh_var[isec + 1][ilat + 1][ip],
01367                        clim_oh_ps[ip + 1],
01368                        clim_oh_var[isec + 1][ilat + 1][ip + 1], p);
01369     aux00 = LIN(clim_oh_lats[ilat], aux00, clim_oh_lats[ilat + 1], aux01, lat);
01370     aux11 = LIN(clim_oh_lats[ilat], aux10, clim_oh_lats[ilat + 1], aux11, lat);
01371     return 1e6 * LIN(clim_oh_secs[isec], aux00,
01372                     clim_oh_secs[isec + 1], aux11, sec);
01373 }

```

Here is the call graph for this function:



5.21.2.5 double clim_tropo (double t, double lat)

Climatology of tropopause pressure.

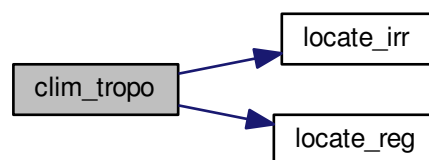
Definition at line 1506 of file libtrac.c.

```

01508         {
01509
01510         /* Get seconds since begin of year... */
01511         double sec = FMOD(t, 365.25 * 86400.);
01512         while (sec < 0)
01513             sec += 365.25 * 86400.;
01514
01515         /* Get indices... */
01516         int isec = locate_irr(clim_tropo_secs, 12, sec);
01517         int ilat = locate_reg(clim_tropo_lats, 73, lat);
01518
01519         /* Interpolate tropopause data (NCEP/NCAR Reanalysis 1)... */
01520         double p0 = LIN(clim_tropo_lats[ilat],
01521                         clim_tropo_tps[isec][ilat],
01522                         clim_tropo_lats[ilat + 1],
01523                         clim_tropo_tps[isec][ilat + 1], lat);
01524         double p1 = LIN(clim_tropo_lats[ilat],
01525                         clim_tropo_tps[isec + 1][ilat],
01526                         clim_tropo_lats[ilat + 1],
01527                         clim_tropo_tps[isec + 1][ilat + 1], lat);
01528         return LIN(clim_tropo_secs[isec], p0, clim_tropo_secs[isec + 1], p1, sec);
01529     }

```

Here is the call graph for this function:



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

Get day of year from date.

Definition at line 1533 of file libtrac.c.

```

01537         {
01538
01539         int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01540         int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01541
01542         /* Get day of year... */
01543         if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
01544             *doy = d0l[mon - 1] + day - 1;
01545         else
01546             *doy = d0[mon - 1] + day - 1;
01547     }

```

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

Get date from day of year.

Definition at line 1551 of file libtrac.c.

```

01555         {
01556
01557     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01558     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
01559     int i;
01560
01561     /* Get month and day... */
01562     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
01563         for (i = 11; i >= 0; i--)
01564             if (d0l[i] <= doy)
01565                 break;
01566         *mon = i + 1;
01567         *day = doy - d0l[i] + 1;
01568     } else {
01569         for (i = 11; i >= 0; i--)
01570             if (d0[i] <= doy)
01571                 break;
01572         *mon = i + 1;
01573         *day = doy - d0[i] + 1;
01574     }
01575 }

```

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

Convert geolocation to Cartesian coordinates.

Definition at line 1579 of file libtrac.c.

```

01583     {
01584
01585     double radius = z + RE;
01586     x[0] = radius * cos(lat / 180. * M_PI) * cos(lon / 180. * M_PI);
01587     x[1] = radius * cos(lat / 180. * M_PI) * sin(lon / 180. * M_PI);
01588     x[2] = radius * sin(lat / 180. * M_PI);
01589 }

```

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

Get meteorological data for given timestep.

Definition at line 1593 of file libtrac.c.

```

01598     {
01599
01600     static int init, ip, ix, iy;
01601
01602     met_t *mets;
01603
01604     char filename[LEN];
01605
01606     /* Init... */
01607     if (t == ctl->t_start || !init) {
01608         init = 1;
01609
01610         get_met_help(t, -1, metbase, ctl->dt_met, filename);
01611         if (!read_met(ctl, filename, *met0))
01612             ERRMSG("Cannot open file!");
01613
01614         get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
dt_met, filename);
01615         if (!read_met(ctl, filename, *met1))
01616             ERRMSG("Cannot open file!");
01617 #ifdef _OPENACC
01618         met_t *met0up = *met0;
01619         met_t *met1up = *met1;
01620 #pragma acc update device(met0up[:1], met1up[:1])
01621 #endif
01622     }
01623
01624     /* Read new data for forward trajectories... */
01625     if (t > (*met1)->time && ctl->direction == 1) {
01626         mets = *met1;
01627         *met1 = *met0;
01628         *met0 = mets;
01629     }
01630 }

```

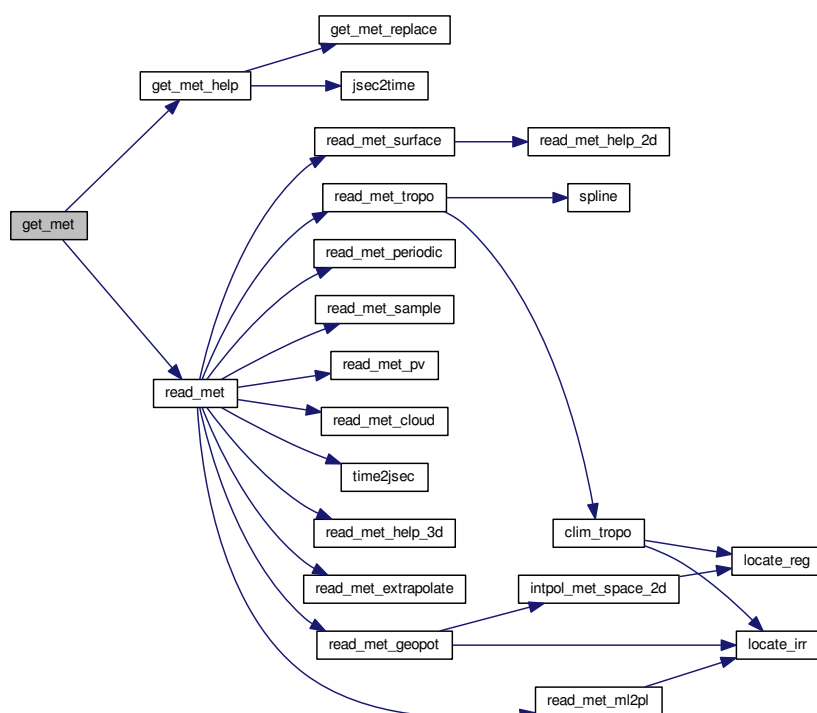


```

01629     get_met_help(t, 1, metbase, ctl->dt_met, filename);
01630     if (!read_met(ctl, filename, *met1))
01631         ERRMSG("Cannot open file!");
01632 #ifdef _OPENACC
01633     met_t *met1up = *met1;
01634     #pragma acc update device(met1up[:1])
01635 #endif
01636 }
01637
01638 /* Read new data for backward trajectories... */
01639 if (t < (*met0)->time && ctl->direction == -1) {
01640     mets = *met1;
01641     *met1 = *met0;
01642     *met0 = mets;
01643     get_met_help(t, -1, metbase, ctl->dt_met, filename);
01644     if (!read_met(ctl, filename, *met0))
01645         ERRMSG("Cannot open file!");
01646 #ifdef _OPENACC
01647     met_t *met0up = *met0;
01648     #pragma acc update device(met0up[:1])
01649 #endif
01650 }
01651
01652 /* Check that grids are consistent... */
01653 if ((*met0)->nx != (*met1)->nx
01654     || (*met0)->ny != (*met1)->ny || (*met0)->np != (*met1)->np)
01655     ERRMSG("Meteo grid dimensions do not match!");
01656 for (ix = 0; ix < (*met0)->nx; ix++)
01657     if ((*met0)->lon[ix] != (*met1)->lon[ix])
01658         ERRMSG("Meteo grid longitudes do not match!");
01659 for (iy = 0; iy < (*met0)->ny; iy++)
01660     if ((*met0)->lat[iy] != (*met1)->lat[iy])
01661         ERRMSG("Meteo grid latitudes do not match!");
01662 for (ip = 0; ip < (*met0)->np; ip++)
01663     if ((*met0)->p[ip] != (*met1)->p[ip])
01664         ERRMSG("Meteo grid pressure levels do not match!");
01665 }

```

Here is the call graph for this function:



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

Get meteorological data for timestep.

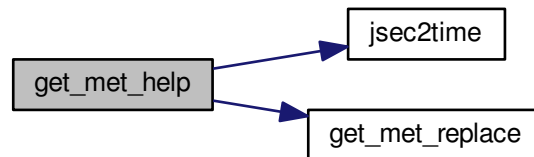
Definition at line 1669 of file libtrac.c.

```

01674         {
01675
01676     char repl[LEN];
01677
01678     double t6, r;
01679
01680     int year, mon, day, hour, min, sec;
01681
01682     /* Round time to fixed intervals... */
01683     if (direct == -1)
01684         t6 = floor(t / dt_met) * dt_met;
01685     else
01686         t6 = ceil(t / dt_met) * dt_met;
01687
01688     /* Decode time... */
01689     jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
01690
01691     /* Set filename... */
01692     sprintf(filename, "%s_YYYY_MM_DD_HH.nc", metbase);
01693     sprintf(repl, "%d", year);
01694     get_met_replace(filename, "YYYY", repl);
01695     sprintf(repl, "%02d", mon);
01696     get_met_replace(filename, "MM", repl);
01697     sprintf(repl, "%02d", day);
01698     get_met_replace(filename, "DD", repl);
01699     sprintf(repl, "%02d", hour);
01700     get_met_replace(filename, "HH", repl);
01701 }

```

Here is the call graph for this function:



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

Replace template strings in filename.

Definition at line 1705 of file libtrac.c.

```

01708         {
01709
01710     char buffer[LEN], *ch;
01711
01712     /* Iterate... */
01713     for (int i = 0; i < 3; i++) {
01714
01715         /* Replace substring... */
01716         if (!(ch = strstr(orig, search)))
01717             return;
01718         strncpy(buffer, orig, (size_t) (ch - orig));
01719         buffer[ch - orig] = 0;
01720         sprintf(buffer + (ch - orig), "%s%s", repl, ch + strlen(search));
01721         orig[0] = 0;
01722         strcpy(orig, buffer);
01723     }
01724 }

```

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

Spatial interpolation of meteorological data.

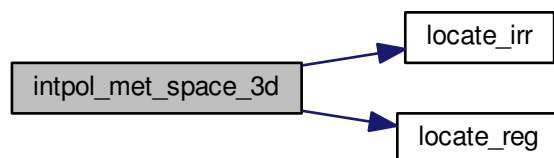
Definition at line 1728 of file [libtrac.c](#).

```

01737         {
01738
01739         /* Check longitude... */
01740         if (met->lon[met->nx - 1] > 180 && lon < 0)
01741             lon += 360;
01742
01743         /* Get interpolation indices and weights... */
01744         if (init) {
01745             ci[0] = locate_irr(met->p, met->np, p);
01746             ci[1] = locate_reg(met->lon, met->nx, lon);
01747             ci[2] = locate_reg(met->lat, met->ny, lat);
01748             cw[0] = (met->p[ci[0] + 1] - p)
01749                 / (met->p[ci[0] + 1] - met->p[ci[0]]);
01750             cw[1] = (met->lon[ci[1] + 1] - lon)
01751                 / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01752             cw[2] = (met->lat[ci[2] + 1] - lat)
01753                 / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01754         }
01755
01756         /* Interpolate vertically... */
01757         double aux00 =
01758             cw[0] * (array[ci[1]][ci[2]][ci[0]] - array[ci[1]][ci[2]][ci[0] + 1])
01759             + array[ci[1]][ci[2]][ci[0] + 1];
01760         double aux01 =
01761             cw[0] * (array[ci[1]][ci[2] + 1][ci[0]] -
01762                 array[ci[1]][ci[2] + 1][ci[0] + 1])
01763             + array[ci[1]][ci[2] + 1][ci[0] + 1];
01764         double aux10 =
01765             cw[0] * (array[ci[1] + 1][ci[2]][ci[0]] -
01766                 array[ci[1] + 1][ci[2]][ci[0] + 1])
01767             + array[ci[1] + 1][ci[2]][ci[0] + 1];
01768         double aux11 =
01769             cw[0] * (array[ci[1] + 1][ci[2] + 1][ci[0]] -
01770                 array[ci[1] + 1][ci[2] + 1][ci[0] + 1])
01771             + array[ci[1] + 1][ci[2] + 1][ci[0] + 1];
01772
01773         /* Interpolate horizontally... */
01774         aux00 = cw[2] * (aux00 - aux01) + aux01;
01775         aux11 = cw[2] * (aux10 - aux11) + aux11;
01776         *var = cw[1] * (aux00 - aux11) + aux11;
01777     }

```

Here is the call graph for this function:



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

Spatial interpolation of meteorological data.

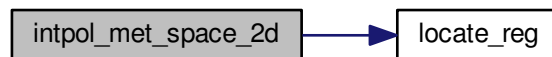
Definition at line 1782 of file [libtrac.c](#).

```

01790         {
01791
01792         /* Check longitude... */
01793         if (met->lon[met->nx - 1] > 180 && lon < 0)
01794             lon += 360;
01795
01796         /* Get interpolation indices and weights... */
01797         if (init) {
01798             ci[1] = locate_reg(met->lon, met->nx, lon);
01799             ci[2] = locate_reg(met->lat, met->ny, lat);
01800             cw[1] = (met->lon[ci[1] + 1] - lon)
01801                 / (met->lon[ci[1] + 1] - met->lon[ci[1]]);
01802             cw[2] = (met->lat[ci[2] + 1] - lat)
01803                 / (met->lat[ci[2] + 1] - met->lat[ci[2]]);
01804         }
01805
01806         /* Set variables... */
01807         double aux00 = array[ci[1]][ci[2]];
01808         double aux01 = array[ci[1]][ci[2] + 1];
01809         double aux10 = array[ci[1] + 1][ci[2]];
01810         double aux11 = array[ci[1] + 1][ci[2] + 1];
01811
01812         /* Interpolate horizontally... */
01813         if (isfinite(aux00) && isfinite(aux01))
01814             aux00 = cw[2] * (aux00 - aux01) + aux01;
01815         else if (cw[2] < 0.5)
01816             aux00 = aux01;
01817         if (isfinite(aux10) && isfinite(aux11))
01818             aux11 = cw[2] * (aux10 - aux11) + aux11;
01819         else if (cw[2] > 0.5)
01820             aux11 = aux10;
01821         if (isfinite(aux00) && isfinite(aux11))
01822             *var = cw[1] * (aux00 - aux11) + aux11;
01823         else {
01824             if (cw[1] > 0.5)
01825                 *var = aux00;
01826             else
01827                 *var = aux11;
01828         }
01829     }

```

Here is the call graph for this function:



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

Temporal interpolation of meteorological data.

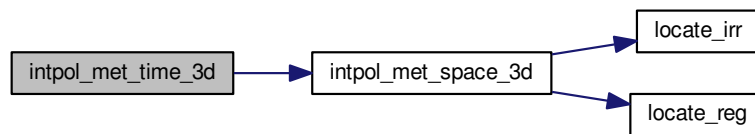
Definition at line 1833 of file libtrac.c.

```

01845         {
01846
01847         double var0, var1, wt;
01848
01849         /* Spatial interpolation... */
01850         intpol_met_space_3d(met0, array0, p, lon, lat, &var0, ci, cw, init);
01851         intpol_met_space_3d(met1, array1, p, lon, lat, &var1, ci, cw, init);
01852
01853         /* Get weighting factor... */
01854         wt = (met1->time - ts) / (met1->time - met0->time);
01855
01856         /* Interpolate... */
01857         *var = wt * (var0 - var1) + var1;
01858     }

```

Here is the call graph for this function:



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

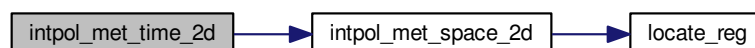
Temporal interpolation of meteorological data.

Definition at line 1862 of file [libtrac.c](#).

```

01873     {
01874
01875     double var0, var1, wt;
01876
01877     /* Spatial interpolation... */
01878     intpol_met_space_2d(met0, array0, lon, lat, &var0, ci, cw, init);
01879     intpol_met_space_2d(met1, array1, lon, lat, &var1, ci, cw, init);
01880
01881     /* Get weighting factor... */
01882     wt = (met1->time - ts) / (met1->time - met0->time);
01883
01884     /* Interpolate... */
01885     *var = wt * (var0 - var1) + var1;
01886 }
  
```

Here is the call graph for this function:



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

Convert seconds to date.

Definition at line 1890 of file [libtrac.c](#).

```

01898     {
01899
01900     struct tm t0, *t1;
01901
01902     t0.tm_year = 100;
01903     t0.tm_mon = 0;
01904     t0.tm_mday = 1;
01905     t0.tm_hour = 0;
  
```

```

01906     t0.tm_min = 0;
01907     t0.tm_sec = 0;
01908
01909     time_t jsec0 = (time_t) jsec + timegm(&t0);
01910     t1 = gmtime(&jsec0);
01911
01912     *year = t1->tm_year + 1900;
01913     *mon = t1->tm_mon + 1;
01914     *day = t1->tm_mday;
01915     *hour = t1->tm_hour;
01916     *min = t1->tm_min;
01917     *sec = t1->tm_sec;
01918     *remain = jsec - floor(jsec);
01919 }

```

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

Find array index for irregular grid.

Definition at line 1923 of file libtrac.c.

```

01926     {
01927
01928     int ilo = 0;
01929     int ihi = n - 1;
01930     int i = (ihi + ilo) >> 1;
01931
01932     if (xx[i] < xx[i + 1])
01933         while (ihi > ilo + 1) {
01934             i = (ihi + ilo) >> 1;
01935             if (xx[i] > x)
01936                 ihi = i;
01937             else
01938                 ilo = i;
01939         } else
01940         while (ihi > ilo + 1) {
01941             i = (ihi + ilo) >> 1;
01942             if (xx[i] <= x)
01943                 ihi = i;
01944             else
01945                 ilo = i;
01946         }
01947
01948     return ilo;
01949 }

```

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

Find array index for regular grid.

Definition at line 1953 of file libtrac.c.

```

01956     {
01957
01958     /* Calculate index... */
01959     int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
01960
01961     /* Check range... */
01962     if (i < 0)
01963         i = 0;
01964     else if (i >= n - 2)
01965         i = n - 2;
01966
01967     return i;
01968 }

```

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

Read atmospheric data.

Definition at line 1972 of file libtrac.c.

```

01975         {
01976
01977     FILE *in;
01978
01979     char line[LEN], *tok;
01980
01981     double t0;
01982
01983     int dimid, ip, iq, ncid, varid;
01984
01985     size_t nparts;
01986
01987     /* Init... */
01988     atm->np = 0;
01989
01990     /* Write info... */
01991     printf("Read atmospheric data: %s\n", filename);
01992
01993     /* Read ASCII data... */
01994     if (ctl->atm_type == 0) {
01995
01996         /* Open file... */
01997         if (!(in = fopen(filename, "r"))) {
01998             WARN("File not found!");
01999             return 0;
02000         }
02001
02002         /* Read line... */
02003         while (fgets(line, LEN, in)) {
02004
02005             /* Read data... */
02006             TOK(line, tok, "%lg", atm->time[atm->np]);
02007             TOK(NULL, tok, "%lg", atm->p[atm->np]);
02008             TOK(NULL, tok, "%lg", atm->lon[atm->np]);
02009             TOK(NULL, tok, "%lg", atm->lat[atm->np]);
02010             for (iq = 0; iq < ctl->nq; iq++)
02011                 TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
02012
02013             /* Convert altitude to pressure... */
02014             atm->p[atm->np] = P(atm->p[atm->np]);
02015
02016             /* Increment data point counter... */
02017             if (++atm->np > NP)
02018                 ERRMSG("Too many data points!");
02019         }
02020
02021         /* Close file... */
02022         fclose(in);
02023     }
02024
02025     /* Read binary data... */
02026     else if (ctl->atm_type == 1) {
02027
02028         /* Open file... */
02029         if (!(in = fopen(filename, "r")))
02030             return 0;
02031
02032         /* Read data... */
02033         FREAD(&atm->np, int, 1, in);
02034         FREAD(atm->time, double,
02035             (size_t) atm->np,
02036             in);
02037         FREAD(atm->p, double,
02038             (size_t) atm->np,
02039             in);
02040         FREAD(atm->lon, double,
02041             (size_t) atm->np,
02042             in);
02043         FREAD(atm->lat, double,
02044             (size_t) atm->np,
02045             in);
02046         for (iq = 0; iq < ctl->nq; iq++)
02047             FREAD(atm->q[iq], double,
02048                 (size_t) atm->np,
02049                 in);
02050
02051         /* Close file... */

```

```

02052     fclose(in);
02053 }
02054
02055 /* Read netCDF data... */
02056 else if (ctl->atm_type == 2) {
02057
02058     /* Open file... */
02059     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR)
02060         return 0;
02061
02062     /* Get dimensions... */
02063     NC(nc_inq_dimid(ncid, "NPARTS", &dimid));
02064     NC(nc_inq_dimlen(ncid, dimid, &nparts));
02065     atm->np = (int) nparts;
02066     if (atm->np > NP)
02067         ERRMSG("Too many particles!");
02068
02069     /* Get time... */
02070     NC(nc_inq_varid(ncid, "time", &varid));
02071     NC(nc_get_var_double(ncid, varid, &t0));
02072     for (ip = 0; ip < atm->np; ip++)
02073         atm->time[ip] = t0;
02074
02075     /* Read geolocations... */
02076     NC(nc_inq_varid(ncid, "PRESS", &varid));
02077     NC(nc_get_var_double(ncid, varid, atm->p));
02078     NC(nc_inq_varid(ncid, "LON", &varid));
02079     NC(nc_get_var_double(ncid, varid, atm->lon));
02080     NC(nc_inq_varid(ncid, "LAT", &varid));
02081     NC(nc_get_var_double(ncid, varid, atm->lat));
02082
02083     /* Read variables... */
02084     if (ctl->qnt_p >= 0)
02085         if (nc_inq_varid(ncid, "PRESS", &varid) == NC_NOERR)
02086             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_p]));
02087     if (ctl->qnt_t >= 0)
02088         if (nc_inq_varid(ncid, "TEMP", &varid) == NC_NOERR)
02089             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_t]));
02090     if (ctl->qnt_u >= 0)
02091         if (nc_inq_varid(ncid, "U", &varid) == NC_NOERR)
02092             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_u]));
02093     if (ctl->qnt_v >= 0)
02094         if (nc_inq_varid(ncid, "V", &varid) == NC_NOERR)
02095             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_v]));
02096     if (ctl->qnt_w >= 0)
02097         if (nc_inq_varid(ncid, "W", &varid) == NC_NOERR)
02098             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_w]));
02099     if (ctl->qnt_h2o >= 0)
02100         if (nc_inq_varid(ncid, "SH", &varid) == NC_NOERR)
02101             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_h2o]));
02102     if (ctl->qnt_o3 >= 0)
02103         if (nc_inq_varid(ncid, "O3", &varid) == NC_NOERR)
02104             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_o3]));
02105     if (ctl->qnt_theta >= 0)
02106         if (nc_inq_varid(ncid, "THETA", &varid) == NC_NOERR)
02107             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_theta]));
02108     if (ctl->qnt_pv >= 0)
02109         if (nc_inq_varid(ncid, "PV", &varid) == NC_NOERR)
02110             NC(nc_get_var_double(ncid, varid, atm->q[ctl->qnt_pv]));
02111
02112     /* Check data... */
02113     for (ip = 0; ip < atm->np; ip++)
02114         if (fabs(atm->lon[ip]) > 360 || fabs(atm->lat[ip]) > 90
02115             || (ctl->qnt_t >= 0 && fabs(atm->q[ctl->qnt_t][ip]) > 350)
02116             || (ctl->qnt_h2o >= 0 && fabs(atm->q[ctl->qnt_h2o][ip]) > 1)
02117             || (ctl->qnt_theta >= 0 && fabs(atm->q[ctl->qnt_theta][ip]) > 1e10)
02118             || (ctl->qnt_pv >= 0 && fabs(atm->q[ctl->qnt_pv][ip]) > 1e10)) {
02119         atm->time[ip] = GSL_NAN;
02120         atm->p[ip] = GSL_NAN;
02121         atm->lon[ip] = GSL_NAN;
02122         atm->lat[ip] = GSL_NAN;
02123         for (iq = 0; iq < ctl->nq; iq++)
02124             atm->q[iq][ip] = GSL_NAN;
02125     } else {
02126         if (ctl->qnt_h2o >= 0)
02127             atm->q[ctl->qnt_h2o][ip] *= 1.608;
02128         if (ctl->qnt_pv >= 0)
02129             atm->q[ctl->qnt_pv][ip] *= 1e6;
02130         if (atm->lon[ip] > 180)
02131             atm->lon[ip] -= 360;
02132     }
02133
02134     /* Close file... */
02135     NC(nc_close(ncid));
02136 }
02137
02138 /* Error... */

```



```

02139     else
02140         ERRMSG("Atmospheric data type not supported!");
02141
02142     /* Check number of points... */
02143     if (atm->np < 1)
02144         ERRMSG("Can not read any data!");
02145
02146     /* Return success... */
02147     return 1;
02148 }

```

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

Read control parameters.

Definition at line 2152 of file libtrac.c.

```

02156     {
02157
02158         /* Write info... */
02159         printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
02160             "(executable: %s | compiled: %s, %s)\n\n",
02161             argv[0], __DATE__, __TIME__);
02162
02163         /* Initialize quantity indices... */
02164         ctl->qnt_ens = -1;
02165         ctl->qnt_m = -1;
02166         ctl->qnt_r = -1;
02167         ctl->qnt_rho = -1;
02168         ctl->qnt_ps = -1;
02169         ctl->qnt_pt = -1;
02170         ctl->qnt_z = -1;
02171         ctl->qnt_p = -1;
02172         ctl->qnt_t = -1;
02173         ctl->qnt_u = -1;
02174         ctl->qnt_v = -1;
02175         ctl->qnt_w = -1;
02176         ctl->qnt_h2o = -1;
02177         ctl->qnt_o3 = -1;
02178         ctl->qnt_lwc = -1;
02179         ctl->qnt_iwc = -1;
02180         ctl->qnt_pc = -1;
02181         ctl->qnt_hno3 = -1;
02182         ctl->qnt_oh = -1;
02183         ctl->qnt_rh = -1;
02184         ctl->qnt_theta = -1;
02185         ctl->qnt_vh = -1;
02186         ctl->qnt_vz = -1;
02187         ctl->qnt_pv = -1;
02188         ctl->qnt_tice = -1;
02189         ctl->qnt_tsts = -1;
02190         ctl->qnt_tnat = -1;
02191         ctl->qnt_stat = -1;
02192
02193         /* Read quantities... */
02194         ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
02195         if (ctl->nq > NQ)
02196             ERRMSG("Too many quantities!");
02197         for (int iq = 0; iq < ctl->nq; iq++) {
02198
02199             /* Read quantity name and format... */
02200             scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
02201             scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
02202                 ctl->qnt_format[iq]);
02203
02204             /* Try to identify quantity... */
02205             if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
02206                 ctl->qnt_ens = iq;
02207                 sprintf(ctl->qnt_unit[iq], "-");
02208             } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
02209                 ctl->qnt_m = iq;
02210                 sprintf(ctl->qnt_unit[iq], "kg");
02211             } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
02212                 ctl->qnt_r = iq;
02213                 sprintf(ctl->qnt_unit[iq], "m");
02214             } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
02215                 ctl->qnt_rho = iq;
02216                 sprintf(ctl->qnt_unit[iq], "kg/m^3");
02217             } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
02218                 ctl->qnt_ps = iq;

```

```

02219     sprintf(ctl->qnt_unit[iq], "hPa");
02220 } else if (strcmp(ctl->qnt_name[iq], "pt") == 0) {
02221     ctl->qnt_pt = iq;
02222     sprintf(ctl->qnt_unit[iq], "hPa");
02223 } else if (strcmp(ctl->qnt_name[iq], "z") == 0) {
02224     ctl->qnt_z = iq;
02225     sprintf(ctl->qnt_unit[iq], "km");
02226 } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
02227     ctl->qnt_p = iq;
02228     sprintf(ctl->qnt_unit[iq], "hPa");
02229 } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
02230     ctl->qnt_t = iq;
02231     sprintf(ctl->qnt_unit[iq], "K");
02232 } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
02233     ctl->qnt_u = iq;
02234     sprintf(ctl->qnt_unit[iq], "m/s");
02235 } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
02236     ctl->qnt_v = iq;
02237     sprintf(ctl->qnt_unit[iq], "m/s");
02238 } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
02239     ctl->qnt_w = iq;
02240     sprintf(ctl->qnt_unit[iq], "hPa/s");
02241 } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
02242     ctl->qnt_h2o = iq;
02243     sprintf(ctl->qnt_unit[iq], "ppv");
02244 } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
02245     ctl->qnt_o3 = iq;
02246     sprintf(ctl->qnt_unit[iq], "ppv");
02247 } else if (strcmp(ctl->qnt_name[iq], "lwc") == 0) {
02248     ctl->qnt_lwc = iq;
02249     sprintf(ctl->qnt_unit[iq], "kg/kg");
02250 } else if (strcmp(ctl->qnt_name[iq], "iwc") == 0) {
02251     ctl->qnt_iwc = iq;
02252     sprintf(ctl->qnt_unit[iq], "kg/kg");
02253 } else if (strcmp(ctl->qnt_name[iq], "pc") == 0) {
02254     ctl->qnt_pc = iq;
02255     sprintf(ctl->qnt_unit[iq], "hPa");
02256 } else if (strcmp(ctl->qnt_name[iq], "hno3") == 0) {
02257     ctl->qnt_hno3 = iq;
02258     sprintf(ctl->qnt_unit[iq], "ppv");
02259 } else if (strcmp(ctl->qnt_name[iq], "oh") == 0) {
02260     ctl->qnt_oh = iq;
02261     sprintf(ctl->qnt_unit[iq], "molec/cm^3");
02262 } else if (strcmp(ctl->qnt_name[iq], "rh") == 0) {
02263     ctl->qnt_rh = iq;
02264     sprintf(ctl->qnt_unit[iq], "%");
02265 } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
02266     ctl->qnt_theta = iq;
02267     sprintf(ctl->qnt_unit[iq], "K");
02268 } else if (strcmp(ctl->qnt_name[iq], "vh") == 0) {
02269     ctl->qnt_vh = iq;
02270     sprintf(ctl->qnt_unit[iq], "m/s");
02271 } else if (strcmp(ctl->qnt_name[iq], "vz") == 0) {
02272     ctl->qnt_vz = iq;
02273     sprintf(ctl->qnt_unit[iq], "m/s");
02274 } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
02275     ctl->qnt_pv = iq;
02276     sprintf(ctl->qnt_unit[iq], "PVU");
02277 } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
02278     ctl->qnt_tice = iq;
02279     sprintf(ctl->qnt_unit[iq], "K");
02280 } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
02281     ctl->qnt_tsts = iq;
02282     sprintf(ctl->qnt_unit[iq], "K");
02283 } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
02284     ctl->qnt_tnat = iq;
02285     sprintf(ctl->qnt_unit[iq], "K");
02286 } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
02287     ctl->qnt_stat = iq;
02288     sprintf(ctl->qnt_unit[iq], "-");
02289 } else
02290     scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
02291 }
02292
02293 /* Time steps of simulation... */
02294 ctl->direction =
02295     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
02296 if (ctl->direction != -1 && ctl->direction != 1)
02297     ERRMSG("Set DIRECTION to -1 or 1!");
02298 ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "1e100", NULL);
02299 ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
02300
02301 /* Meteorological data... */
02302 ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
02303 ctl->met_dx = (int) scan_ctl(filename, argc, argv, "MET_DX", -1, "1", NULL);
02304 ctl->met_dy = (int) scan_ctl(filename, argc, argv, "MET_DY", -1, "1", NULL);
02305 ctl->met_dp = (int) scan_ctl(filename, argc, argv, "MET_DP", -1, "1", NULL);

```

```

02306   ctl->met_sx = (int) scan_ctl(filename, argc, argv, "MET_SX", -1, "1", NULL);
02307   ctl->met_sy = (int) scan_ctl(filename, argc, argv, "MET_SY", -1, "1", NULL);
02308   ctl->met_sp = (int) scan_ctl(filename, argc, argv, "MET_SP", -1, "1", NULL);
02309   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
02310   if (ctl->met_np > EP)
02311       ERRMSG("Too many levels!");
02312   for (int ip = 0; ip < ctl->met_np; ip++)
02313       ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
02314   ctl->met_tropo =
02315       (int) scan_ctl(filename, argc, argv, "MET_TROPO", -1, "3", NULL);
02316   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
02317   ctl->met_dt_out =
02318       scan_ctl(filename, argc, argv, "MET_DT_OUT", -1, "0.1", NULL);
02319
02320   /* Isosurface parameters... */
02321   ctl->isosurf =
02322       (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
02323   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
02324
02325   /* Diffusion parameters... */
02326   ctl->turb_dx_trop =
02327       scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50", NULL);
02328   ctl->turb_dx_strat =
02329       scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0", NULL);
02330   ctl->turb_dz_trop =
02331       scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0", NULL);
02332   ctl->turb_dz_strat =
02333       scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
02334   ctl->turb_mesox =
02335       scan_ctl(filename, argc, argv, "TURB_MESOX", -1, "0.16", NULL);
02336   ctl->turb_mesoz =
02337       scan_ctl(filename, argc, argv, "TURB_MESOZ", -1, "0.16", NULL);
02338
02339   /* Species parameters... */
02340   scan_ctl(filename, argc, argv, "SPECIES", -1, "-", ctl->species);
02341   if (strcmp(ctl->species, "SO2") == 0) {
02342       ctl->molmass = 64.066;
02343       ctl->oh_chem[0] = 3.3e-31; /* (JPL Publication 15-10) */
02344       ctl->oh_chem[1] = 4.3; /* (JPL Publication 15-10) */
02345       ctl->oh_chem[2] = 1.6e-12; /* (JPL Publication 15-10) */
02346       ctl->oh_chem[3] = 0.0; /* (JPL Publication 15-10) */
02347       ctl->wet_depo[0] = 2.0e-05; /* (FLEXPART v10.4) */
02348       ctl->wet_depo[1] = 0.62; /* (FLEXPART v10.4) */
02349       ctl->wet_depo[2] = 1.3e-2; /* (Sander, 2015) */
02350       ctl->wet_depo[3] = 2900.0; /* (Sander, 2015) */
02351   } else {
02352       ctl->molmass =
02353           scan_ctl(filename, argc, argv, "MOLMASS", -1, "-999", NULL);
02354       ctl->tdec_trop =
02355           scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
02356       ctl->tdec_strat =
02357           scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
02358       for (int ip = 0; ip < 4; ip++)
02359           ctl->oh_chem[ip] =
02360               scan_ctl(filename, argc, argv, "OH_CHEM", ip, "0", NULL);
02361       for (int ip = 0; ip < 4; ip++)
02362           ctl->wet_depo[ip] =
02363               scan_ctl(filename, argc, argv, "WET_DEPO", ip, "0", NULL);
02364   }
02365
02366   /* PSC analysis... */
02367   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
02368   ctl->psc_hno3 =
02369       scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
02370
02371   /* Output of atmospheric data... */
02372   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
atm_basename);
02373   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
02374   ctl->atm_dt_out =
02375       scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
02376   ctl->atm_filter =
02377       (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
02378   ctl->atm_stride =
02379       (int) scan_ctl(filename, argc, argv, "ATM_STRIDE", -1, "1", NULL);
02380   ctl->atm_type =
02381       (int) scan_ctl(filename, argc, argv, "ATM_TYPE", -1, "0", NULL);
02382
02383   /* Output of CSI data... */
02384   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
csi_basename);
02385   ctl->csi_dt_out =
02386       scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
02387   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "-", ctl->
csi_obsfile);
02388   ctl->csi_obsmin =
02389       scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);

```

```

02390 ctl->csi_modmin =
02391     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
02392 ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
02393 ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
02394 ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
02395 ctl->csi_lon0 =
02396     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
02397 ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
02398 ctl->csi_nx =
02399     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
02400 ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
02401 ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
02402 ctl->csi_ny =
02403     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
02404
02405 /* Output of ensemble data... */
02406 scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
ens_basename);
02407
02408 /* Output of grid data... */
02409 scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
02410     ctl->grid_basename);
02411 scan_ctl(filename, argc, argv, "GRID_GPPFILE", -1, "-", ctl->
grid_gppfile);
02412 ctl->grid_dt_out =
02413     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
02414 ctl->grid_sparse =
02415     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
02416 ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
02417 ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
02418 ctl->grid_nz =
02419     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
02420 ctl->grid_lon0 =
02421     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
02422 ctl->grid_lon1 =
02423     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
02424 ctl->grid_nx =
02425     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
02426 ctl->grid_lat0 =
02427     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
02428 ctl->grid_lat1 =
02429     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
02430 ctl->grid_ny =
02431     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
02432
02433 /* Output of profile data... */
02434 scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
02435     ctl->prof_basename);
02436 scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
prof_obsfile);
02437 ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
02438 ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
02439 ctl->prof_nz =
02440     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
02441 ctl->prof_lon0 =
02442     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
02443 ctl->prof_lon1 =
02444     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
02445 ctl->prof_nx =
02446     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
02447 ctl->prof_lat0 =
02448     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
02449 ctl->prof_lat1 =
02450     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
02451 ctl->prof_ny =
02452     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
02453
02454 /* Output of station data... */
02455 scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
02456     ctl->stat_basename);
02457 ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
02458 ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
02459 ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
02460 }

```

Here is the call graph for this function:



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

Read meteorological data file.

Definition at line 2464 of file [libtrac.c](#).

```

02467         {
02468
02469     char cmd[2 * LEN], levname[LEN], tstr[10];
02470
02471     int ip, dimid, ncid, varid, year, mon, day, hour;
02472
02473     size_t np, nx, ny;
02474
02475     /* Write info... */
02476     printf("Read meteorological data: %s\n", filename);
02477
02478     /* Get time from filename... */
02479     sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
02480     year = atoi(tstr);
02481     sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
02482     mon = atoi(tstr);
02483     sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
02484     day = atoi(tstr);
02485     sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
02486     hour = atoi(tstr);
02487     time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
02488
02489     /* Open netCDF file... */
02490     if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02491
02492         /* Try to stage meteo file... */
02493         if (ctl->met_stage[0] != '-') {
02494             sprintf(cmd, "%s %d %02d %02d %s", ctl->met_stage,
02495                 year, mon, day, hour, filename);
02496             if (system(cmd) != 0)
02497                 ERRMSG("Error while staging meteo data!");
02498         }
02499
02500         /* Try to open again... */
02501         if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
02502             WARN("File not found!");
02503             return 0;
02504         }
02505     }
02506
02507     /* Get dimensions... */
02508     NC(nc_inq_dimid(ncid, "lon", &dimid));
02509     NC(nc_inq_dimlen(ncid, dimid, &nx));
02510     if (nx < 2 || nx > EX)
02511         ERRMSG("Number of longitudes out of range!");
02512
02513     NC(nc_inq_dimid(ncid, "lat", &dimid));
02514     NC(nc_inq_dimlen(ncid, dimid, &ny));
02515     if (ny < 2 || ny > EY)
02516         ERRMSG("Number of latitudes out of range!");
02517
02518     sprintf(levname, "lev");
02519     NC(nc_inq_dimid(ncid, levname, &dimid));
02520     NC(nc_inq_dimlen(ncid, dimid, &np));
02521     if (np == 1) {
02522         sprintf(levname, "lev_2");
02523         if (nc_inq_dimid(ncid, levname, &dimid) != NC_NOERR) {

```

```

02524     sprintf(levname, "plev");
02525     nc_inq_dimid(ncid, levname, &dimid);
02526 }
02527 NC(nc_inq_dimlen(ncid, dimid, &np));
02528 }
02529 if (np < 2 || np > EP)
02530     ERRMSG("Number of levels out of range!");
02531
02532 /* Store dimensions... */
02533 met->np = (int) np;
02534 met->nx = (int) nx;
02535 met->ny = (int) ny;
02536
02537 /* Get horizontal grid... */
02538 NC(nc_inq_varid(ncid, "lon", &varid));
02539 NC(nc_get_var_double(ncid, varid, met->lon));
02540 NC(nc_inq_varid(ncid, "lat", &varid));
02541 NC(nc_get_var_double(ncid, varid, met->lat));
02542
02543 /* Read meteorological data... */
02544 if (!read_met_help_3d(ncid, "t", "T", met, met->t, 1.0))
02545     ERRMSG("Cannot read temperature!");
02546 if (!read_met_help_3d(ncid, "u", "U", met, met->u, 1.0))
02547     ERRMSG("Cannot read zonal wind!");
02548 if (!read_met_help_3d(ncid, "v", "V", met, met->v, 1.0))
02549     ERRMSG("Cannot read meridional wind!");
02550 if (!read_met_help_3d(ncid, "w", "W", met, met->w, 0.01f))
02551     WARN("Cannot read vertical velocity");
02552 if (!read_met_help_3d(ncid, "q", "Q", met, met->h2o, (float) (MA / MH2O)))
02553     WARN("Cannot read specific humidity!");
02554 if (!read_met_help_3d(ncid, "o3", "O3", met, met->o3, (float) (MA / MO3)))
02555     WARN("Cannot read ozone data!");
02556 if (!read_met_help_3d(ncid, "clwc", "CLWC", met, met->lwc, 1.0))
02557     WARN("Cannot read cloud liquid water content!");
02558 if (!read_met_help_3d(ncid, "ciwc", "CIWC", met, met->iwc, 1.0))
02559     WARN("Cannot read cloud ice water content!");
02560
02561 /* Meteo data on pressure levels... */
02562 if (ctl->met_np <= 0) {
02563
02564     /* Read pressure levels from file... */
02565     NC(nc_inq_varid(ncid, levname, &varid));
02566     NC(nc_get_var_double(ncid, varid, met->p));
02567     for (ip = 0; ip < met->np; ip++)
02568         met->p[ip] /= 100.;
02569
02570     /* Extrapolate data for lower boundary... */
02571     read_met_extrapolate(met);
02572 }
02573
02574 /* Meteo data on model levels... */
02575 else {
02576
02577     /* Read pressure data from file... */
02578     read_met_help_3d(ncid, "pl", "PL", met, met->p, 0.01f);
02579
02580     /* Interpolate from model levels to pressure levels... */
02581     read_met_ml2pl(ctl, met, met->t);
02582     read_met_ml2pl(ctl, met, met->u);
02583     read_met_ml2pl(ctl, met, met->v);
02584     read_met_ml2pl(ctl, met, met->w);
02585     read_met_ml2pl(ctl, met, met->h2o);
02586     read_met_ml2pl(ctl, met, met->o3);
02587     read_met_ml2pl(ctl, met, met->lwc);
02588     read_met_ml2pl(ctl, met, met->iwc);
02589
02590     /* Set pressure levels... */
02591     met->np = ctl->met_np;
02592     for (ip = 0; ip < met->np; ip++)
02593         met->p[ip] = ctl->met_p[ip];
02594 }
02595
02596 /* Check ordering of pressure levels... */
02597 for (ip = 1; ip < met->np; ip++)
02598     if (met->p[ip - 1] < met->p[ip])
02599         ERRMSG("Pressure levels must be descending!");
02600
02601 /* Read surface data... */
02602 read_met_surface(ncid, met);
02603
02604 /* Create periodic boundary conditions... */
02605 read_met_periodic(met);
02606
02607 /* Downsampling... */
02608 read_met_sample(ctl, met);
02609
02610 /* Calculate geopotential heights... */

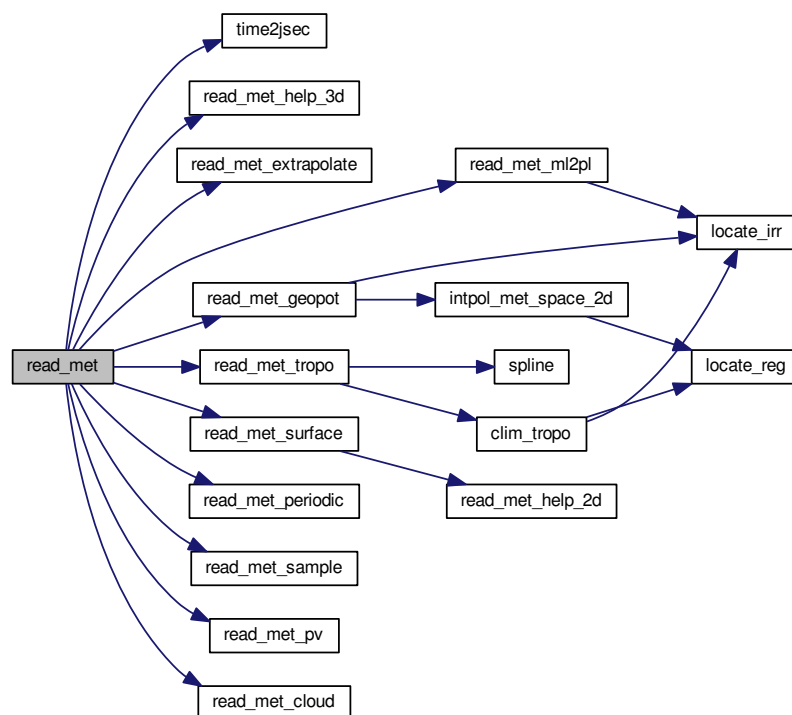
```

```

02611  read_met_geopot (met);
02612
02613  /* Calculate potential vorticity... */
02614  read_met_pv (met);
02615
02616  /* Calculate tropopause pressure... */
02617  read_met_tropo (ctl, met);
02618
02619  /* Calculate cloud properties... */
02620  read_met_cloud (met);
02621
02622  /* Close file... */
02623  NC(nc_close(ncid));
02624
02625  /* Return success... */
02626  return 1;
02627 }

```

Here is the call graph for this function:



5.21.2.22 void read_met_cloud (met_t * met)

Calculate cloud properties.

Definition at line 2631 of file libtrac.c.

```

02632  {
02633
02634  int ix, iy, ip;
02635
02636  /* Loop over columns... */
02637  #pragma omp parallel for default(shared) private(ix,iy,ip)
02638  for (ix = 0; ix < met->nx; ix++)
02639  for (iy = 0; iy < met->ny; iy++) {
02640

```

```

02641      /* Init... */
02642      met->pc[ix][iy] = GSL_NAN;
02643      met->cl[ix][iy] = 0;
02644
02645      /* Loop over pressure levels... */
02646      for (ip = 0; ip < met->np - 1; ip++) {
02647
02648          /* Check pressure... */
02649          if (met->p[ip] > met->ps[ix][iy] || met->p[ip] < P(20.))
02650              continue;
02651
02652          /* Get cloud top pressure ... */
02653          if (met->iwc[ix][iy][ip] > 0 || met->lwc[ix][iy][ip] > 0)
02654              met->pc[ix][iy] = (float) met->p[ip + 1];
02655
02656          /* Get cloud water... */
02657          met->cl[ix][iy] += (float)
02658              (0.5 * (met->iwc[ix][iy][ip] + met->iwc[ix][iy][ip + 1]
02659                  + met->lwc[ix][iy][ip] + met->lwc[ix][iy][ip + 1])
02660               * 100. * (met->p[ip] - met->p[ip + 1]) / G0);
02661      }
02662  }
02663 }

```

5.21.2.23 void read_met_extrapolate (met_t * met)

Extrapolate meteorological data at lower boundary.

Definition at line 2667 of file libtrac.c.

```

02668      {
02669
02670          int ip, ip0, ix, iy;
02671
02672          /* Loop over columns... */
02673          #pragma omp parallel for default(shared) private(ix,iy,ip0,ip)
02674          for (ix = 0; ix < met->nx; ix++)
02675              for (iy = 0; iy < met->ny; iy++) {
02676
02677                  /* Find lowest valid data point... */
02678                  for (ip0 = met->np - 1; ip0 >= 0; ip0--)
02679                      if (!isfinite(met->t[ix][iy][ip0])
02680                          || !isfinite(met->u[ix][iy][ip0])
02681                          || !isfinite(met->v[ix][iy][ip0])
02682                          || !isfinite(met->w[ix][iy][ip0]))
02683                      break;
02684
02685                  /* Extrapolate... */
02686                  for (ip = ip0; ip >= 0; ip--) {
02687                      met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
02688                      met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
02689                      met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
02690                      met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
02691                      met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
02692                      met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
02693                      met->lwc[ix][iy][ip] = met->lwc[ix][iy][ip + 1];
02694                      met->iwc[ix][iy][ip] = met->iwc[ix][iy][ip + 1];
02695                  }
02696              }
02697      }

```

5.21.2.24 void read_met_geopot (met_t * met)

Calculate geopotential heights.

Definition at line 2701 of file libtrac.c.

```

02702      {
02703
02704          const int dx = 6, dy = 4;
02705
02706          static float help[EX][EY][EP];
02707
02708          double logp[EP], ts, z0, cw[3];

```

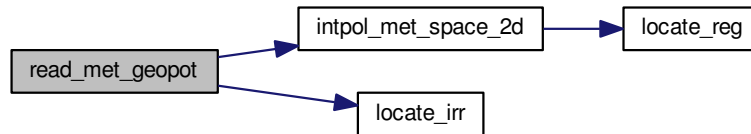


```

02709
02710     int ip, ip0, ix, ix2, ix3, iy, iy2, n, ci[3];
02711
02712     /* Calculate log pressure... */
02713     for (ip = 0; ip < met->np; ip++)
02714         logp[ip] = log(met->p[ip]);
02715
02716     /* Initialize geopotential heights... */
02717     #pragma omp parallel for default(shared) private(ix,iy,ip)
02718     for (ix = 0; ix < met->nx; ix++)
02719         for (iy = 0; iy < met->ny; iy++)
02720             for (ip = 0; ip < met->np; ip++)
02721                 met->z[ix][iy][ip] = GSL_NAN;
02722
02723     /* Apply hydrostatic equation to calculate geopotential heights... */
02724     #pragma omp parallel for default(shared) private(ix,iy,z0,ip0,ts,ip,ci,cw)
02725     for (ix = 0; ix < met->nx; ix++)
02726         for (iy = 0; iy < met->ny; iy++) {
02727
02728             /* Get surface height... */
02729             intpol_met_space_2d(met, met->z0, met->lon[ix], met->
lat[iy], &z0, ci,
                                cw, 1);
02730
02731
02732             /* Find surface pressure level index... */
02733             ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);
02734
02735             /* Get virtual temperature at the surface... */
02736             ts =
02737                 LIN(met->p[ip0],
02738                     TVIRT(met->t[ix][iy][ip0], met->h2o[ix][iy][ip0]),
02739                     met->p[ip0 + 1],
02740                     TVIRT(met->t[ix][iy][ip0 + 1], met->h2o[ix][iy][ip0 + 1]),
02741                     met->ps[ix][iy]);
02742
02743             /* Upper part of profile... */
02744             met->z[ix][iy][ip0 + 1]
02745                 = (float) (z0 + RI / MA / G0 * 0.5
02746                     * (ts + TVIRT(met->t[ix][iy][ip0 + 1],
02747                         met->h2o[ix][iy][ip0 + 1]))
02748                     * (log(met->ps[ix][iy]) - logp[ip0 + 1]));
02749             for (ip = ip0 + 2; ip < met->np; ip++)
02750                 met->z[ix][iy][ip]
02751                     = (float) (met->z[ix][iy][ip - 1] + RI / MA / G0 * 0.5 *
02752                         (TVIRT(met->t[ix][iy][ip - 1], met->h2o[ix][iy][ip - 1])
02753                             + TVIRT(met->t[ix][iy][ip], met->h2o[ix][iy][ip]))
02754                         * (logp[ip - 1] - logp[ip]));
02755         }
02756
02757     /* Horizontal smoothing... */
02758     #pragma omp parallel for default(shared) private(ix,iy,ip,n,ix2,ix3,iy2)
02759     for (ix = 0; ix < met->nx; ix++)
02760         for (iy = 0; iy < met->ny; iy++)
02761             for (ip = 0; ip < met->np; ip++) {
02762                 n = 0;
02763                 help[ix][iy][ip] = 0;
02764                 for (ix2 = ix - dx; ix2 <= ix + dx; ix2++) {
02765                     ix3 = ix2;
02766                     if (ix3 < 0)
02767                         ix3 += met->nx;
02768                     else if (ix3 >= met->nx)
02769                         ix3 -= met->nx;
02770                     for (iy2 = GSL_MAX(iy - dy, 0);
02771                         iy2 <= GSL_MIN(iy + dy, met->ny - 1); iy2++)
02772                         if (isfinite(met->z[ix3][iy2][ip])) {
02773                             help[ix][iy][ip] += met->z[ix3][iy2][ip];
02774                             n++;
02775                         }
02776                 }
02777                 if (n > 0)
02778                     help[ix][iy][ip] /= (float) n;
02779                 else
02780                     help[ix][iy][ip] = GSL_NAN;
02781             }
02782
02783     /* Copy data... */
02784     #pragma omp parallel for default(shared) private(ix,iy,ip)
02785     for (ix = 0; ix < met->nx; ix++)
02786         for (iy = 0; iy < met->ny; iy++)
02787             for (ip = 0; ip < met->np; ip++)
02788                 met->z[ix][iy][ip] = help[ix][iy][ip];
02789 }

```

Here is the call graph for this function:



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

Read and convert 3D variable from meteorological data file.

Definition at line 2793 of file libtrac.c.

```

02799     {
02800
02801     float *help;
02802
02803     int ip, ix, iy, varid;
02804
02805     /* Check if variable exists... */
02806     if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02807         if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02808             return 0;
02809
02810     /* Allocate... */
02811     ALLOC(help, float, EX * EY * EP);
02812
02813     /* Read data... */
02814     NC(nc_get_var_float(ncid, varid, help));
02815
02816     /* Copy and check data... */
02817     #pragma omp parallel for default(shared) private(ix,iy,ip)
02818     for (ix = 0; ix < met->nx; ix++)
02819         for (iy = 0; iy < met->ny; iy++)
02820             for (ip = 0; ip < met->np; ip++) {
02821                 dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
02822                 if (fabsf(dest[ix][iy][ip]) < 1e14f)
02823                     dest[ix][iy][ip] *= scl;
02824                 else
02825                     dest[ix][iy][ip] = GSL_NAN;
02826             }
02827
02828     /* Free... */
02829     free(help);
02830
02831     /* Return... */
02832     return 1;
02833 }
  
```

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

Read and convert 2D variable from meteorological data file.

Definition at line 2837 of file libtrac.c.

```

02843     {
02844
02845     float *help;
02846
02847     int ix, iy, varid;
02848
  
```

```

02849  /* Check if variable exists... */
02850  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
02851      if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
02852          return 0;
02853
02854  /* Allocate... */
02855  ALLOC(help, float, EX * EY);
02856
02857  /* Read data... */
02858  NC(nc_get_var_float(ncid, varid, help));
02859
02860  /* Copy and check data... */
02861  #pragma omp parallel for default(shared) private(ix,iy)
02862  for (ix = 0; ix < met->nx; ix++)
02863      for (iy = 0; iy < met->ny; iy++) {
02864          dest[ix][iy] = help[iy * met->nx + ix];
02865          if (fabsf(dest[ix][iy]) < 1e14f)
02866              dest[ix][iy] *= scl;
02867          else
02868              dest[ix][iy] = GSL_NAN;
02869      }
02870
02871  /* Free... */
02872  free(help);
02873
02874  /* Return... */
02875  return 1;
02876 }

```

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

Convert meteorological data from model levels to pressure levels.

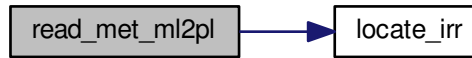
Definition at line 2880 of file libtrac.c.

```

02883      {
02884
02885      double aux[EP], p[EP], pt;
02886
02887      int ip, ip2, ix, iy;
02888
02889      /* Loop over columns... */
02890      #pragma omp parallel for default(shared) private(ix,iy,ip,p,pt,ip2,aux)
02891      for (ix = 0; ix < met->nx; ix++)
02892          for (iy = 0; iy < met->ny; iy++) {
02893
02894              /* Copy pressure profile... */
02895              for (ip = 0; ip < met->np; ip++)
02896                  p[ip] = met->pl[ix][iy][ip];
02897
02898              /* Interpolate... */
02899              for (ip = 0; ip < ctl->met_np; ip++) {
02900                  pt = ctl->met_p[ip];
02901                  if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
02902                      pt = p[0];
02903                  else if ((pt > p[met->np - 1] && p[1] > p[0])
02904                      || (pt < p[met->np - 1] && p[1] < p[0]))
02905                      pt = p[met->np - 1];
02906                  ip2 = locate_irr(p, met->np, pt);
02907                  aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
02908                      p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
02909              }
02910
02911              /* Copy data... */
02912              for (ip = 0; ip < ctl->met_np; ip++)
02913                  var[ix][iy][ip] = (float) aux[ip];
02914          }
02915      }

```

Here is the call graph for this function:



5.21.2.28 void read_met_periodic (met_t * met)

Create meteorological data with periodic boundary conditions.

Definition at line 2919 of file libtrac.c.

```

02920         {
02921
02922         /* Check longitudes... */
02923         if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
02924                 + met->lon[1] - met->lon[0] - 360) < 0.01))
02925             return;
02926
02927         /* Increase longitude counter... */
02928         if ((++met->nx) > EX)
02929             ERRMSG("Cannot create periodic boundary conditions!");
02930
02931         /* Set longitude... */
02932         met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
lon[0];
02933
02934         /* Loop over latitudes and pressure levels... */
02935 #pragma omp parallel for default(shared)
02936         for (int iy = 0; iy < met->ny; iy++) {
02937             met->ps[met->nx - 1][iy] = met->ps[0][iy];
02938             met->zs[met->nx - 1][iy] = met->zs[0][iy];
02939             for (int ip = 0; ip < met->np; ip++) {
02940                 met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
02941                 met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
02942                 met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
02943                 met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
02944                 met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
02945                 met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
02946                 met->lwc[met->nx - 1][iy][ip] = met->lwc[0][iy][ip];
02947                 met->iwc[met->nx - 1][iy][ip] = met->iwc[0][iy][ip];
02948             }
02949         }
02950     }
  
```

5.21.2.29 void read_met_pv (met_t * met)

Calculate potential vorticity.

Definition at line 2954 of file libtrac.c.

```

02955         {
02956
02957         double c0, c1, cr, dx, dy, dp0, dp1, denom, dtdx, dvdx, dtdy, dudy,
02958             dtdp, dudp, dvdp, latr, vort, pows[EP];
02959
02960         int ip, ip0, ip1, ix, ix0, ix1, iy, iy0, iy1;
02961
02962         /* Set powers... */
02963         for (ip = 0; ip < met->np; ip++)
02964             pows[ip] = pow(1000. / met->p[ip], 0.286);
02965     }
  
```

```

02966  /* Loop over grid points... */
02967 #pragma omp parallel for default(shared)
02968     private(ix,ix0,ix1,iy,iy0,iy1,latr,dx,dy,c0,c1,cr,vort,ip,ip0,ip1,dp0,dp1,denom,dt dx,dvdx,dt dy,dudy,dt dp,dudp,dvdp)
02969     for (ix = 0; ix < met->nx; ix++) {
02970         /* Set indices... */
02971         ix0 = GSL_MAX(ix - 1, 0);
02972         ix1 = GSL_MIN(ix + 1, met->nx - 1);
02973
02974         /* Loop over grid points... */
02975         for (iy = 0; iy < met->ny; iy++) {
02976             /* Set indices... */
02977             iy0 = GSL_MAX(iy - 1, 0);
02978             iy1 = GSL_MIN(iy + 1, met->ny - 1);
02979
02980             /* Set auxiliary variables... */
02981             latr = 0.5 * (met->lat[iy1] + met->lat[iy0]);
02982             dx = 1000. * DEG2DX(met->lon[ix1] - met->lon[ix0], latr);
02983             dy = 1000. * DEG2DY(met->lat[iy1] - met->lat[iy0]);
02984             c0 = cos(met->lat[iy0] / 180. * M_PI);
02985             c1 = cos(met->lat[iy1] / 180. * M_PI);
02986             cr = cos(latr / 180. * M_PI);
02987             vort = 2 * 7.2921e-5 * sin(latr * M_PI / 180.);
02988
02989             /* Loop over grid points... */
02990             for (ip = 0; ip < met->np; ip++) {
02991                 /* Get gradients in longitude... */
02992                 dt dx = (met->t[ix1][iy][ip] - met->t[ix0][iy][ip]) * pows[ip] / dx;
02993                 dv dx = (met->v[ix1][iy][ip] - met->v[ix0][iy][ip]) / dx;
02994
02995                 /* Get gradients in latitude... */
02996                 dt dy = (met->t[ix][iy1][ip] - met->t[ix][iy0][ip]) * pows[ip] / dy;
02997                 dudy = (met->u[ix][iy1][ip] * c1 - met->u[ix][iy0][ip] * c0) / dy;
02998
02999                 /* Set indices... */
03000                 ip0 = GSL_MAX(ip - 1, 0);
03001                 ip1 = GSL_MIN(ip + 1, met->np - 1);
03002
03003                 /* Get gradients in pressure... */
03004                 dp0 = 100. * (met->p[ip] - met->p[ip0]);
03005                 dp1 = 100. * (met->p[ip1] - met->p[ip]);
03006                 if (ip != ip0 && ip != ip1) {
03007                     denom = dp0 * dp1 * (dp0 + dp1);
03008                     dt dp = (dp0 * dp0 * met->t[ix][iy][ip1] * pows[ip1]
03009                             - dp1 * dp1 * met->t[ix][iy][ip0] * pows[ip0]
03010                             + (dp1 * dp1 - dp0 * dp0) * met->t[ix][iy][ip] * pows[ip])
03011                             / denom;
03012                     dudp = (dp0 * dp0 * met->u[ix][iy][ip1]
03013                             - dp1 * dp1 * met->u[ix][iy][ip0]
03014                             + (dp1 * dp1 - dp0 * dp0) * met->u[ix][iy][ip])
03015                             / denom;
03016                     dv dp = (dp0 * dp0 * met->v[ix][iy][ip1]
03017                             - dp1 * dp1 * met->v[ix][iy][ip0]
03018                             + (dp1 * dp1 - dp0 * dp0) * met->v[ix][iy][ip])
03019                             / denom;
03020                 } else {
03021                     denom = dp0 + dp1;
03022                     dt dp =
03023                         (met->t[ix][iy][ip1] * pows[ip1] -
03024                         met->t[ix][iy][ip0] * pows[ip0]) / denom;
03025                     dudp = (met->u[ix][iy][ip1] - met->u[ix][iy][ip0]) / denom;
03026                     dv dp = (met->v[ix][iy][ip1] - met->v[ix][iy][ip0]) / denom;
03027                 }
03028
03029                 /* Calculate PV... */
03030                 met->pv[ix][iy][ip] = (float)
03031                     (1e6 * G0 *
03032                     (-dt dp * (dv dx - dudy / cr + vort) + dv dp * dt dx - dudp * dt dy));
03033             }
03034         }
03035     }
03036 }
03037 }
03038
03039 /* Fix for polar regions... */
03040 #pragma omp parallel for default(shared) private(ix,ip)
03041     for (ix = 0; ix < met->nx; ix++)
03042         for (ip = 0; ip < met->np; ip++) {
03043             met->pv[ix][0][ip]
03044                 = met->pv[ix][1][ip]
03045                 = met->pv[ix][2][ip];
03046             met->pv[ix][met->ny - 1][ip]
03047                 = met->pv[ix][met->ny - 2][ip]
03048                 = met->pv[ix][met->ny - 3][ip];
03049         }
03050 }

```

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

Downsampling of meteorological data.

Definition at line 3054 of file libtrac.c.

```

03056         {
03057
03058     met_t *help;
03059
03060     float w, wsum;
03061
03062     int ip, ip2, ix, ix2, ix3, iy, iy2;
03063
03064     /* Check parameters... */
03065     if (ctl->met_dp <= 1 && ctl->met_dx <= 1 && ctl->met_dy <= 1
03066         && ctl->met_sp <= 1 && ctl->met_sx <= 1 && ctl->met_sy <= 1)
03067         return;
03068
03069     /* Allocate... */
03070     ALLOC(help, met_t, 1);
03071
03072     /* Copy data... */
03073     help->nx = met->nx;
03074     help->ny = met->ny;
03075     help->np = met->np;
03076     memcpy(help->lon, met->lon, sizeof(met->lon));
03077     memcpy(help->lat, met->lat, sizeof(met->lat));
03078     memcpy(help->p, met->p, sizeof(met->p));
03079
03080     /* Smoothing... */
03081     for (ix = 0; ix < met->nx; ix += ctl->met_dx) {
03082         for (iy = 0; iy < met->ny; iy += ctl->met_dy) {
03083             for (ip = 0; ip < met->np; ip += ctl->met_dp) {
03084                 help->ps[ix][iy] = 0;
03085                 help->zs[ix][iy] = 0;
03086                 help->t[ix][iy][ip] = 0;
03087                 help->u[ix][iy][ip] = 0;
03088                 help->v[ix][iy][ip] = 0;
03089                 help->w[ix][iy][ip] = 0;
03090                 help->h2o[ix][iy][ip] = 0;
03091                 help->o3[ix][iy][ip] = 0;
03092                 help->lwc[ix][iy][ip] = 0;
03093                 help->iwc[ix][iy][ip] = 0;
03094                 wsum = 0;
03095                 for (ix2 = ix - ctl->met_sx + 1; ix2 <= ix + ctl->met_sx - 1; ix2++) {
03096                     ix3 = ix2;
03097                     if (ix3 < 0)
03098                         ix3 += met->nx;
03099                     else if (ix3 >= met->nx)
03100                         ix3 -= met->nx;
03101
03102                     for (iy2 = GSL_MAX(iy - ctl->met_sy + 1, 0);
03103                         iy2 <= GSL_MIN(iy + ctl->met_sy - 1, met->ny - 1); iy2++)
03104                         for (ip2 = GSL_MAX(ip - ctl->met_sp + 1, 0);
03105                             ip2 <= GSL_MIN(ip + ctl->met_sp - 1, met->np - 1); ip2++) {
03106                             w = (float) (1.0 - fabs(ix - ix2) / ctl->met_sx)
03107                                 * (float) (1.0 - fabs(iy - iy2) / ctl->met_sy)
03108                                 * (float) (1.0 - fabs(ip - ip2) / ctl->met_dp);
03109                             help->ps[ix][iy] += w * met->ps[ix3][iy2];
03110                             help->zs[ix][iy] += w * met->zs[ix3][iy2];
03111                             help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
03112                             help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
03113                             help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
03114                             help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
03115                             help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
03116                             help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
03117                             help->lwc[ix][iy][ip] += w * met->lwc[ix3][iy2][ip2];
03118                             help->iwc[ix][iy][ip] += w * met->iwc[ix3][iy2][ip2];
03119                             wsum += w;
03120                         }
03121                     }
03122                 help->ps[ix][iy] /= wsum;
03123                 help->zs[ix][iy] /= wsum;
03124                 help->t[ix][iy][ip] /= wsum;
03125                 help->u[ix][iy][ip] /= wsum;
03126                 help->v[ix][iy][ip] /= wsum;
03127                 help->w[ix][iy][ip] /= wsum;
03128                 help->h2o[ix][iy][ip] /= wsum;
03129                 help->o3[ix][iy][ip] /= wsum;
03130                 help->lwc[ix][iy][ip] /= wsum;
03131                 help->iwc[ix][iy][ip] /= wsum;
03132             }
03133         }
03134     }

```

```

03133     }
03134 }
03135
03136 /* Downsampling... */
03137 met->nx = 0;
03138 for (ix = 0; ix < help->nx; ix += ctl->met_dx) {
03139     met->lon[met->nx] = help->lon[ix];
03140     met->ny = 0;
03141     for (iy = 0; iy < help->ny; iy += ctl->met_dy) {
03142         met->lat[met->ny] = help->lat[iy];
03143         met->ps[met->nx][met->ny] = help->ps[ix][iy];
03144         met->zs[met->nx][met->ny] = help->zs[ix][iy];
03145         met->np = 0;
03146         for (ip = 0; ip < help->np; ip += ctl->met_dp) {
03147             met->p[met->np] = help->p[ip];
03148             met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
03149             met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
03150             met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
03151             met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
03152             met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
03153             met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
03154             met->lwc[met->nx][met->ny][met->np] = help->lwc[ix][iy][ip];
03155             met->iwc[met->nx][met->ny][met->np] = help->iwc[ix][iy][ip];
03156             met->np++;
03157         }
03158         met->ny++;
03159     }
03160     met->nx++;
03161 }
03162
03163 /* Free... */
03164 free(help);
03165 }

```

5.21.2.31 void read_met_surface (int ncid, met_t * met)

Read surface data.

Definition at line 3169 of file libtrac.c.

```

03171     {
03172
03173     int ix, iy;
03174
03175     /* Read surface pressure... */
03176     if (!read_met_help_2d(ncid, "ps", "PS", met, met->ps, 0.01f)) {
03177         if (!read_met_help_2d(ncid, "lnsp", "LNSP", met, met->ps, 1.0)) {
03178             ERRMSG("Cannot not read surface pressure data!");
03179             for (ix = 0; ix < met->nx; ix++)
03180                 for (iy = 0; iy < met->ny; iy++)
03181                     met->ps[ix][iy] = (float) met->p[0];
03182         } else {
03183             for (iy = 0; iy < met->ny; iy++)
03184                 for (ix = 0; ix < met->nx; ix++)
03185                     met->ps[ix][iy] = (float) (exp(met->ps[ix][iy]) / 100.);
03186         }
03187     }
03188
03189     /* Read geopotential height at the surface... */
03190     if (!read_met_help_2d
03191         (ncid, "z", "Z", met, met->zs, (float) (1. / (1000. * G0))))
03192         if (!read_met_help_2d
03193             (ncid, "zm", "ZM", met, met->zs, (float) (1. / 1000.)))
03194             ERRMSG("Cannot read surface geopotential height!");
03195 }

```

Here is the call graph for this function:



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

Calculate tropopause pressure.

Definition at line 3199 of file libtrac.c.

```

03201         {
03202
03203     double p2[200], pv[EP], pv2[200], t[EP], t2[200], th[EP],
03204           th2[200], z[EP], z2[200];
03205
03206     int found, ix, iy, iz, iz2;
03207
03208     /* Get altitude and pressure profiles... */
03209     for (iz = 0; iz < met->np; iz++)
03210         z[iz] = Z(met->p[iz]);
03211     for (iz = 0; iz <= 190; iz++) {
03212         z2[iz] = 4.5 + 0.1 * iz;
03213         p2[iz] = P(z2[iz]);
03214     }
03215
03216     /* Do not calculate tropopause... */
03217     if (ctl->met_tropo == 0)
03218         for (ix = 0; ix < met->nx; ix++)
03219             for (iy = 0; iy < met->ny; iy++)
03220                 met->pt[ix][iy] = GSL_NAN;
03221
03222     /* Use tropopause climatology... */
03223     else if (ctl->met_tropo == 1) {
03224 #pragma omp parallel for default(shared) private(ix,iy)
03225         for (ix = 0; ix < met->nx; ix++)
03226             for (iy = 0; iy < met->ny; iy++)
03227                 met->pt[ix][iy] = (float) clim_tropo(met->time, met->lat[iy]);
03228     }
03229
03230     /* Use cold point... */
03231     else if (ctl->met_tropo == 2) {
03232
03233         /* Loop over grid points... */
03234 #pragma omp parallel for default(shared) private(ix,iy,iz,t,t2)
03235         for (ix = 0; ix < met->nx; ix++)
03236             for (iy = 0; iy < met->ny; iy++) {
03237
03238                 /* Interpolate temperature profile... */
03239                 for (iz = 0; iz < met->np; iz++)
03240                     t[iz] = met->t[ix][iy][iz];
03241                 spline(z, t, met->np, z2, t2, 171);
03242
03243                 /* Find minimum... */
03244                 iz = (int) gsl_stats_min_index(t2, 1, 171);
03245                 if (iz > 0 && iz < 170)
03246                     met->pt[ix][iy] = (float) p2[iz];
03247                 else
03248                     met->pt[ix][iy] = GSL_NAN;
03249             }
03250     }
03251
03252     /* Use WMO definition... */
03253     else if (ctl->met_tropo == 3 || ctl->met_tropo == 4) {
03254
03255         /* Loop over grid points... */
03256 #pragma omp parallel for default(shared) private(ix,iy,iz,iz2,t,t2,found)
03257         for (ix = 0; ix < met->nx; ix++)
03258             for (iy = 0; iy < met->ny; iy++) {
03259
03260                 /* Interpolate temperature profile... */
03261                 for (iz = 0; iz < met->np; iz++)
03262                     t[iz] = met->t[ix][iy][iz];
03263                 spline(z, t, met->np, z2, t2, 191);
03264
03265                 /* Find 1st tropopause... */
03266                 met->pt[ix][iy] = GSL_NAN;
03267                 for (iz = 0; iz <= 170; iz++) {
03268                     found = 1;
03269                     for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03270                         if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03271                             * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03272                             found = 0;
03273                             break;
03274                         }
03275                 }
03276                 if (found) {
03277                     if (iz > 0 && iz < 170)
03278                         met->pt[ix][iy] = (float) p2[iz];
03279                 }
03280             }
03281     }
03282 }

```

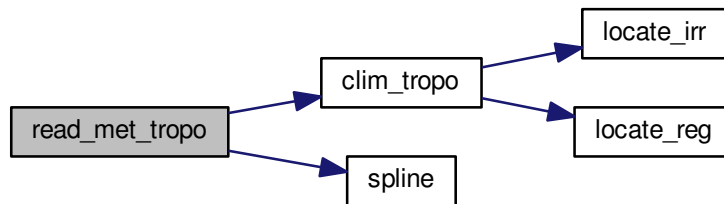


```

03278         break;
03279     }
03280 }
03281
03282 /* Find 2nd tropopause... */
03283 if (ctl->met_tropo == 4) {
03284     met->pt[ix][iy] = GSL_NAN;
03285     for (; iz <= 170; iz++) {
03286         found = 1;
03287         for (iz2 = iz + 1; iz2 <= iz + 10; iz2++)
03288             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03289                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) < 3.0) {
03290                 found = 0;
03291                 break;
03292             }
03293         if (found)
03294             break;
03295     }
03296     for (; iz <= 170; iz++) {
03297         found = 1;
03298         for (iz2 = iz + 1; iz2 <= iz + 20; iz2++)
03299             if (1e3 * G0 / RA * (t2[iz2] - t2[iz]) / (t2[iz2] + t2[iz])
03300                 * (p2[iz2] + p2[iz]) / (p2[iz2] - p2[iz]) > 2.0) {
03301                 found = 0;
03302                 break;
03303             }
03304         if (found) {
03305             if (iz > 0 && iz < 170)
03306                 met->pt[ix][iy] = (float) p2[iz];
03307             break;
03308         }
03309     }
03310 }
03311 }
03312 }
03313
03314 /* Use dynamical tropopause... */
03315 else if (ctl->met_tropo == 5) {
03316
03317     /* Loop over grid points... */
03318     #pragma omp parallel for default(shared) private(ix,iy,iz,pv,pv2,th,th2)
03319     for (ix = 0; ix < met->nx; ix++)
03320         for (iy = 0; iy < met->ny; iy++) {
03321
03322             /* Interpolate potential vorticity profile... */
03323             for (iz = 0; iz < met->np; iz++)
03324                 pv[iz] = met->pv[ix][iy][iz];
03325             spline(z, pv, met->np, z2, pv2, 171);
03326
03327             /* Interpolate potential temperature profile... */
03328             for (iz = 0; iz < met->np; iz++)
03329                 th[iz] = THETA(met->p[iz], met->t[ix][iy][iz]);
03330             spline(z, th, met->np, z2, th2, 171);
03331
03332             /* Find dynamical tropopause 3.5 PVU + 380 K */
03333             met->pt[ix][iy] = GSL_NAN;
03334             for (iz = 0; iz <= 170; iz++)
03335                 if (fabs(pv2[iz]) >= 3.5 || th2[iz] >= 380.) {
03336                     if (iz > 0 && iz < 170)
03337                         met->pt[ix][iy] = (float) p2[iz];
03338                     break;
03339                 }
03340         }
03341     }
03342
03343     else
03344         ERRMSG("Cannot calculate tropopause!");
03345 }

```

Here is the call graph for this function:



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

Read a control parameter from file or command line.

Definition at line 3349 of file `libtrac.c`.

```

03356         {
03357
03358     FILE *in = NULL;
03359
03360     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
03361         msg[2 * LEN], rvarname[LEN], rval[LEN];
03362
03363     int contain = 0, i;
03364
03365     /* Open file... */
03366     if (filename[strlen(filename) - 1] != '-')
03367         if (!(in = fopen(filename, "r")))
03368             ERRMSG("Cannot open file!");
03369
03370     /* Set full variable name... */
03371     if (arridx >= 0) {
03372         sprintf(fullname1, "%s[%d]", varname, arridx);
03373         sprintf(fullname2, "%s[*]", varname);
03374     } else {
03375         sprintf(fullname1, "%s", varname);
03376         sprintf(fullname2, "%s", varname);
03377     }
03378
03379     /* Read data... */
03380     if (in != NULL)
03381         while (fgets(line, LEN, in))
03382             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
03383                 if (strcasecmp(rvarname, fullname1) == 0 ||
03384                     strcasecmp(rvarname, fullname2) == 0) {
03385                     contain = 1;
03386                     break;
03387                 }
03388     for (i = 1; i < argc - 1; i++)
03389         if (strcasecmp(argv[i], fullname1) == 0 ||
03390             strcasecmp(argv[i], fullname2) == 0) {
03391             sprintf(rval, "%s", argv[i + 1]);
03392             contain = 1;
03393             break;
03394         }
03395
03396     /* Close file... */
03397     if (in != NULL)
03398         fclose(in);
03399
03400     /* Check for missing variables... */
03401     if (!contain) {
03402         if (strlen(defvalue) > 0)
03403             sprintf(rval, "%s", defvalue);

```

```

03404     else {
03405         sprintf(msg, "Missing variable %s!\n", fullname1);
03406         ERRMSG(msg);
03407     }
03408 }
03409
03410 /* Write info... */
03411 printf("%s = %s\n", fullname1, rval);
03412
03413 /* Return values... */
03414 if (value != NULL)
03415     sprintf(value, "%s", rval);
03416 return atof(rval);
03417 }

```

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

Spline interpolation.

Definition at line 3421 of file libtrac.c.

```

03427     {
03428
03429         gsl_interp_accel *acc;
03430
03431         gsl_spline *s;
03432
03433         /* Allocate... */
03434         acc = gsl_interp_accel_alloc();
03435         s = gsl_spline_alloc(gsl_interp_cspline, (size_t) n);
03436
03437         /* Interpolate temperature profile... */
03438         gsl_spline_init(s, x, y, (size_t) n);
03439         for (int i = 0; i < n2; i++)
03440             if (x2[i] <= x[0])
03441                 y2[i] = y[0];
03442             else if (x2[i] >= x[n - 1])
03443                 y2[i] = y[n - 1];
03444             else
03445                 y2[i] = gsl_spline_eval(s, x2[i], acc);
03446
03447         /* Free... */
03448         gsl_spline_free(s);
03449         gsl_interp_accel_free(acc);
03450     }

```

5.21.2.35 double stddev (double * data, int n)

Calculate standard deviation.

Definition at line 3454 of file libtrac.c.

```

03456     {
03457
03458         if (n <= 0)
03459             return 0;
03460
03461         double avg = 0, rms = 0;
03462
03463         for (int i = 0; i < n; ++i)
03464             avg += data[i];
03465         avg /= n;
03466
03467         for (int i = 0; i < n; ++i)
03468             rms += SQR(data[i] - avg);
03469
03470         return sqrt(rms / (n - 1));
03471     }

```

5.21.2.36 void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double * jsec)

Convert date to seconds.

Definition at line 3475 of file libtrac.c.

```

03483         {
03484
03485     struct tm t0, t1;
03486
03487     t0.tm_year = 100;
03488     t0.tm_mon = 0;
03489     t0.tm_mday = 1;
03490     t0.tm_hour = 0;
03491     t0.tm_min = 0;
03492     t0.tm_sec = 0;
03493
03494     t1.tm_year = year - 1900;
03495     t1.tm_mon = mon - 1;
03496     t1.tm_mday = day;
03497     t1.tm_hour = hour;
03498     t1.tm_min = min;
03499     t1.tm_sec = sec;
03500
03501     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
03502 }
```

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

Measure wall-clock time.

Definition at line 3506 of file libtrac.c.

```

03509         {
03510
03511     static double starttime[NTIMER], runtime[NTIMER];
03512
03513     /* Check id... */
03514     if (id < 0 || id >= NTIMER)
03515         ERRMSG("Too many timers!");
03516
03517     /* Start timer... */
03518     if (mode == 1) {
03519         if (starttime[id] <= 0)
03520             starttime[id] = omp_get_wtime();
03521         else
03522             ERRMSG("Timer already started!");
03523     }
03524
03525     /* Stop timer... */
03526     else if (mode == 2) {
03527         if (starttime[id] > 0) {
03528             runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
03529             starttime[id] = -1;
03530         }
03531     }
03532
03533     /* Print timer... */
03534     else if (mode == 3) {
03535         printf("%s = %.3f s\n", name, runtime[id]);
03536         runtime[id] = 0;
03537     }
03538 }
```

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

Write atmospheric data.

Definition at line 3542 of file libtrac.c.

```

03546     {
03547
03548     FILE *in, *out;
03549
03550     char line[LEN];
03551
03552     double r, t0, t1;
03553
03554     int ip, iq, year, mon, day, hour, min, sec;
03555
03556     /* Set time interval for output... */
03557     t0 = t - 0.5 * ctl->dt_mod;
03558     t1 = t + 0.5 * ctl->dt_mod;
03559
03560     /* Write info... */
03561     printf("Write atmospheric data: %s\n", filename);
03562
03563     /* Write ASCII data... */
03564     if (ctl->atm_type == 0) {
03565
03566         /* Check if gnuplot output is requested... */
03567         if (ctl->atm_gpfile[0] != '-') {
03568
03569             /* Create gnuplot pipe... */
03570             if (!(out = popen("gnuplot", "w")))
03571                 ERRMSG("Cannot create pipe to gnuplot!");
03572
03573             /* Set plot filename... */
03574             fprintf(out, "set out \"%s.png\"\n", filename);
03575
03576             /* Set time string... */
03577             jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
03578             fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
03579                     year, mon, day, hour, min);
03580
03581             /* Dump gnuplot file to pipe... */
03582             if (!(in = fopen(ctl->atm_gpfile, "r")))
03583                 ERRMSG("Cannot open file!");
03584             while (fgets(line, LEN, in))
03585                 fprintf(out, "%s", line);
03586             fclose(in);
03587         }
03588     else {
03589
03590         /* Create file... */
03591         if (!(out = fopen(filename, "w")))
03592             ERRMSG("Cannot create file!");
03593     }
03594
03595     /* Write header... */
03596     fprintf(out,
03597            "# $1 = time [s]\n"
03598            "# $2 = altitude [km]\n"
03599            "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03600     for (iq = 0; iq < ctl->nq; iq++)
03601         fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
03602                ctl->qnt_unit[iq]);
03603     fprintf(out, "\n");
03604
03605     /* Write data... */
03606     for (ip = 0; ip < atm->np; ip += ctl->atm_stride) {
03607
03608         /* Check time... */
03609         if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
03610             continue;
03611
03612         /* Write output... */
03613         fprintf(out, "%.2f %g %g", atm->time[ip], Z(atm->p[ip]),
03614                atm->lon[ip], atm->lat[ip]);
03615         for (iq = 0; iq < ctl->nq; iq++) {
03616             fprintf(out, " ");
03617             fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
03618         }
03619         fprintf(out, "\n");
03620     }
03621 }
03622

```

```

03623     /* Close file... */
03624     fclose(out);
03625 }
03626
03627 /* Write binary data... */
03628 else if (ctl->atm_type == 1) {
03629
03630     /* Create file... */
03631     if (!(out = fopen(filename, "w")))
03632         ERRMSG("Cannot create file!");
03633
03634     /* Write data... */
03635     FWRITE(&atm->np, int,
03636           1,
03637           out);
03638     FWRITE(atm->time, double,
03639           (size_t) atm->np,
03640           out);
03641     FWRITE(atm->p, double,
03642           (size_t) atm->np,
03643           out);
03644     FWRITE(atm->lon, double,
03645           (size_t) atm->np,
03646           out);
03647     FWRITE(atm->lat, double,
03648           (size_t) atm->np,
03649           out);
03650     for (iq = 0; iq < ctl->nq; iq++)
03651         FWRITE(atm->q[iq], double,
03652               (size_t) atm->np,
03653               out);
03654
03655     /* Close file... */
03656     fclose(out);
03657 }
03658
03659 /* Error... */
03660 else
03661     ERRMSG("Atmospheric data type not supported!");
03662 }

```

Here is the call graph for this function:



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

Write CSI data.

Definition at line 3666 of file libtrac.c.

```

03670     {
03671
03672     static FILE *in, *out;
03673
03674     static char line[LEN];
03675
03676     static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
03677         rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
03678
03679     static int obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
03680
03681     /* Init... */
03682     if (t == ctl->t_start) {

```

```

03683
03684 /* Check quantity index for mass... */
03685 if (ctl->qnt_m < 0)
03686     ERRMSG("Need quantity mass!");
03687
03688 /* Open observation data file... */
03689 printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
03690 if (!(in = fopen(ctl->csi_obsfile, "r")))
03691     ERRMSG("Cannot open file!");
03692
03693 /* Create new file... */
03694 printf("Write CSI data: %s\n", filename);
03695 if (!(out = fopen(filename, "w")))
03696     ERRMSG("Cannot create file!");
03697
03698 /* Write header... */
03699 fprintf(out,
03700     "# $1 = time [s]\n"
03701     "# $2 = number of hits (cx)\n"
03702     "# $3 = number of misses (cy)\n"
03703     "# $4 = number of false alarms (cz)\n"
03704     "# $5 = number of observations (cx + cy)\n"
03705     "# $6 = number of forecasts (cx + cz)\n"
03706     "# $7 = bias (forecasts/observations) [%%]\n"
03707     "# $8 = probability of detection (POD) [%%]\n"
03708     "# $9 = false alarm rate (FAR) [%%]\n"
03709     "# $10 = critical success index (CSI) [%%]\n\n");
03710 }
03711
03712 /* Set time interval... */
03713 t0 = t - 0.5 * ctl->dt_mod;
03714 t1 = t + 0.5 * ctl->dt_mod;
03715
03716 /* Initialize grid cells... */
03717 #pragma omp parallel for default(shared) private(ix,iy,iz)
03718 for (ix = 0; ix < ctl->csi_nx; ix++)
03719     for (iy = 0; iy < ctl->csi_ny; iy++)
03720         for (iz = 0; iz < ctl->csi_nz; iz++)
03721             modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
03722
03723 /* Read observation data... */
03724 while (fgets(line, LEN, in)) {
03725
03726     /* Read data... */
03727     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
03728         5)
03729         continue;
03730
03731     /* Check time... */
03732     if (rt < t0)
03733         continue;
03734     if (rt > t1)
03735         break;
03736
03737     /* Calculate indices... */
03738     ix = (int) ((rlon - ctl->csi_lon0)
03739         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03740     iy = (int) ((rlat - ctl->csi_lat0)
03741         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03742     iz = (int) ((rz - ctl->csi_z0)
03743         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
03744
03745     /* Check indices... */
03746     if (ix < 0 || ix >= ctl->csi_nx ||
03747         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03748         continue;
03749
03750     /* Get mean observation index... */
03751     obsmean[ix][iy][iz] += robs;
03752     obscount[ix][iy][iz]++;
03753 }
03754
03755 /* Analyze model data... */
03756 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
03757 for (ip = 0; ip < atm->np; ip++) {
03758
03759     /* Check time... */
03760     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03761         continue;
03762
03763     /* Get indices... */
03764     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
03765         / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
03766     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
03767         / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
03768     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
03769         / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);

```

```

03770
03771     /* Check indices... */
03772     if (ix < 0 || ix >= ctl->csi_nx ||
03773         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
03774         continue;
03775
03776     /* Get total mass in grid cell... */
03777     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
03778 }
03779
03780 /* Analyze all grid cells... */
03781 #pragma omp parallel for default(shared) private(ix,iy,iz,dlon,dlat,lat,area)
03782 for (ix = 0; ix < ctl->csi_nx; ix++)
03783     for (iy = 0; iy < ctl->csi_ny; iy++)
03784         for (iz = 0; iz < ctl->csi_nz; iz++) {
03785
03786             /* Calculate mean observation index... */
03787             if (obscount[ix][iy][iz] > 0)
03788                 obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
03789
03790             /* Calculate column density... */
03791             if (modmean[ix][iy][iz] > 0) {
03792                 dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
03793                 dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
03794                 lat = ctl->csi_lat0 + dlat * (iy + 0.5);
03795                 area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
03796                     * cos(lat * M_PI / 180.);
03797                 modmean[ix][iy][iz] /= (1e6 * area);
03798             }
03799
03800             /* Calculate CSI... */
03801             if (obscount[ix][iy][iz] > 0) {
03802                 if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03803                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03804                     cx++;
03805                 else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
03806                     modmean[ix][iy][iz] < ctl->csi_modmin)
03807                     cy++;
03808                 else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
03809                     modmean[ix][iy][iz] >= ctl->csi_modmin)
03810                     cz++;
03811             }
03812         }
03813
03814     /* Write output... */
03815     if (fmod(t, ctl->csi_dt_out) == 0) {
03816
03817         /* Write... */
03818         fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
03819             t, cx, cy, cz, cx + cy, cx + cz,
03820             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
03821             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
03822             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
03823             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
03824
03825         /* Set counters to zero... */
03826         cx = cy = cz = 0;
03827     }
03828
03829     /* Close file... */
03830     if (t == ctl->t_stop)
03831         fclose(out);
03832 }

```

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

Write ensemble data.

Definition at line 3836 of file libtrac.c.

```

03840     {
03841
03842         static FILE *out;
03843
03844         static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
03845             t0, t1, x[NENS][3], xm[3];
03846
03847         static int ip, iq;
03848
03849         static size_t i, n;

```



```

03850
03851 /* Init... */
03852 if (t == ctl->t_start) {
03853
03854     /* Check quantities... */
03855     if (ctl->qnt_ens < 0)
03856         ERRMSG("Missing ensemble IDs!");
03857
03858     /* Create new file... */
03859     printf("Write ensemble data: %s\n", filename);
03860     if (!(out = fopen(filename, "w")))
03861         ERRMSG("Cannot create file!");
03862
03863     /* Write header... */
03864     fprintf(out,
03865             "# $1 = time [s]\n"
03866             "# $2 = altitude [km]\n"
03867             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
03868     for (iq = 0; iq < ctl->nq; iq++)
03869         fprintf(out, "# $d = %s (mean) [%s]\n", 5 + iq,
03870                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03871     for (iq = 0; iq < ctl->nq; iq++)
03872         fprintf(out, "# $d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
03873                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
03874     fprintf(out, "# $d = number of members\n\n", 5 + 2 * ctl->nq);
03875 }
03876
03877 /* Set time interval... */
03878 t0 = t - 0.5 * ctl->dt_mod;
03879 t1 = t + 0.5 * ctl->dt_mod;
03880
03881 /* Init... */
03882 ens = GSL_NAN;
03883 n = 0;
03884
03885 /* Loop over air parcels... */
03886 for (ip = 0; ip < atm->np; ip++) {
03887
03888     /* Check time... */
03889     if (atm->time[ip] < t0 || atm->time[ip] > t1)
03890         continue;
03891
03892     /* Check ensemble id... */
03893     if (atm->q[ctl->qnt_ens][ip] != ens) {
03894
03895         /* Write results... */
03896         if (n > 0) {
03897
03898             /* Get mean position... */
03899             xm[0] = xm[1] = xm[2] = 0;
03900             for (i = 0; i < n; i++) {
03901                 xm[0] += x[i][0] / (double) n;
03902                 xm[1] += x[i][1] / (double) n;
03903                 xm[2] += x[i][2] / (double) n;
03904             }
03905             cart2geo(xm, &dummy, &lon, &lat);
03906             fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
03907                     lat);
03908
03909             /* Get quantity statistics... */
03910             for (iq = 0; iq < ctl->nq; iq++) {
03911                 fprintf(out, " ");
03912                 fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03913             }
03914             for (iq = 0; iq < ctl->nq; iq++) {
03915                 fprintf(out, " ");
03916                 fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03917             }
03918             fprintf(out, " %lu\n", n);
03919         }
03920
03921         /* Init new ensemble... */
03922         ens = atm->q[ctl->qnt_ens][ip];
03923         n = 0;
03924     }
03925
03926     /* Save data... */
03927     p[n] = atm->p[ip];
03928     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
03929     for (iq = 0; iq < ctl->nq; iq++)
03930         q[iq][n] = atm->q[iq][ip];
03931     if ((++n) >= NENS)
03932         ERRMSG("Too many data points!");
03933 }
03934
03935 /* Write results... */
03936 if (n > 0) {

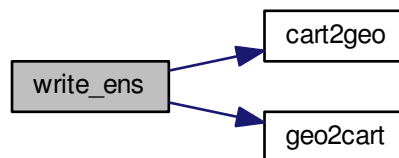
```

```

03937
03938     /* Get mean position... */
03939     xm[0] = xm[1] = xm[2] = 0;
03940     for (i = 0; i < n; i++) {
03941         xm[0] += x[i][0] / (double) n;
03942         xm[1] += x[i][1] / (double) n;
03943         xm[2] += x[i][2] / (double) n;
03944     }
03945     cart2geo(xm, &dummy, &lon, &lat);
03946     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
03947
03948     /* Get quantity statistics... */
03949     for (iq = 0; iq < ctl->nq; iq++) {
03950         fprintf(out, " ");
03951         fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
03952     }
03953     for (iq = 0; iq < ctl->nq; iq++) {
03954         fprintf(out, " ");
03955         fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
03956     }
03957     fprintf(out, " %lu\n", n);
03958 }
03959
03960 /* Close file... */
03961 if (t == ctl->t_stop)
03962     fclose(out);
03963 }

```

Here is the call graph for this function:



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

Write gridded data.

Definition at line 3967 of file `libtrac.c`.

```

03973     {
03974
03975     FILE *in, *out;
03976
03977     char line[LEN];
03978
03979     static double mass[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
03980         area, rho_air, press, temp, cd, vmr, t0, t1, r, cw[3];
03981
03982     static int ip, ix, iy, iz, np[GX][GY][GZ], year, mon, day, hour, min, sec,
03983         ci[3];
03984
03985     /* Check dimensions... */
03986     if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
03987         ERRMSG("Grid dimensions too large!");
03988
03989     /* Set time interval for output... */
03990     t0 = t - 0.5 * ctl->dt_mod;
03991     t1 = t + 0.5 * ctl->dt_mod;
03992
03993     /* Set grid box size... */
03994     dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;

```

```

03995     dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
03996     dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
03997
03998     /* Initialize grid... */
03999 #pragma omp parallel for default(shared) private(ix,iy,iz)
04000     for (ix = 0; ix < ctl->grid_nx; ix++)
04001         for (iy = 0; iy < ctl->grid_ny; iy++)
04002             for (iz = 0; iz < ctl->grid_nz; iz++) {
04003                 mass[ix][iy][iz] = 0;
04004                 np[ix][iy][iz] = 0;
04005             }
04006
04007     /* Average data... */
04008 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04009     for (ip = 0; ip < atm->np; ip++)
04010         if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
04011
04012             /* Get index... */
04013             ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
04014             iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
04015             iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
04016
04017             /* Check indices... */
04018             if (ix < 0 || ix >= ctl->grid_nx ||
04019                 iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
04020                 continue;
04021
04022             /* Add mass... */
04023             if (ctl->qnt_m >= 0)
04024                 mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04025             np[ix][iy][iz]++;
04026         }
04027
04028     /* Check if gnuplot output is requested... */
04029     if (ctl->grid_gpfile[0] != '-') {
04030
04031         /* Write info... */
04032         printf("Plot grid data: %s.png\n", filename);
04033
04034         /* Create gnuplot pipe... */
04035         if (!(out = popen("gnuplot", "w")))
04036             ERRMSG("Cannot create pipe to gnuplot!");
04037
04038         /* Set plot filename... */
04039         fprintf(out, "set out \"%s.png\"\n", filename);
04040
04041         /* Set time string... */
04042         jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
04043         fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
04044             year, mon, day, hour, min);
04045
04046         /* Dump gnuplot file to pipe... */
04047         if (!(in = fopen(ctl->grid_gpfile, "r")))
04048             ERRMSG("Cannot open file!");
04049         while (fgets(line, LEN, in))
04050             fprintf(out, "%s", line);
04051         fclose(in);
04052     }
04053
04054     else {
04055
04056         /* Write info... */
04057         printf("Write grid data: %s\n", filename);
04058
04059         /* Create file... */
04060         if (!(out = fopen(filename, "w")))
04061             ERRMSG("Cannot create file!");
04062     }
04063
04064     /* Write header... */
04065     fprintf(out,
04066         "# $1 = time [s]\n"
04067         "# $2 = altitude [km]\n"
04068         "# $3 = longitude [deg]\n"
04069         "# $4 = latitude [deg]\n"
04070         "# $5 = surface area [km^2]\n"
04071         "# $6 = layer width [km]\n"
04072         "# $7 = number of particles [l]\n"
04073         "# $8 = column density [kg/m^2]\n"
04074         "# $9 = volume mixing ratio [ppv]\n\n");
04075
04076     /* Write data... */
04077     for (ix = 0; ix < ctl->grid_nx; ix++) {
04078         if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
04079             fprintf(out, "\n");
04080         for (iy = 0; iy < ctl->grid_ny; iy++) {
04081             if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)

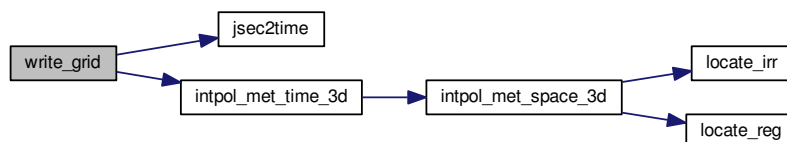
```

```

04082     fprintf(out, "\n");
04083     for (iz = 0; iz < ctl->grid_nz; iz++)
04084     if (!ctl->grid_sparse || mass[ix][iy][iz] > 0) {
04085
04086         /* Set coordinates... */
04087         z = ctl->grid_z0 + dz * (iz + 0.5);
04088         lon = ctl->grid_lon0 + dlon * (ix + 0.5);
04089         lat = ctl->grid_lat0 + dlat * (iy + 0.5);
04090
04091         /* Get pressure and temperature... */
04092         press = P(z);
04093         intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04094             lat, &temp, ci, cw, 1);
04095
04096         /* Calculate surface area... */
04097         area = dlat * dlon * SQR(RE * M_PI / 180.)
04098             * cos(lat * M_PI / 180.);
04099
04100         /* Calculate column density... */
04101         cd = mass[ix][iy][iz] / (1e6 * area);
04102
04103         /* Calculate volume mixing ratio... */
04104         rho_air = 100. * press / (RA * temp);
04105         vmr = (ctl->molmass > 0) ? MA / ctl->molmass * mass[ix][iy][iz]
04106             / (rho_air * 1e6 * area * 1e3 * dz) : GSL_NAN;
04107
04108         /* Write output... */
04109         fprintf(out, "%.2f %g %g %g %g %g %d %g %g\n",
04110             t, z, lon, lat, area, dz, np[ix][iy][iz], cd, vmr);
04111     }
04112 }
04113 }
04114
04115 /* Close file... */
04116 fclose(out);
04117 }

```

Here is the call graph for this function:



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

Write profile data.

Definition at line 4121 of file libtrac.c.

```

04127     {
04128
04129     static FILE *in, *out;
04130
04131     static char line[LEN];
04132
04133     static double mass[GX][GY][GZ], obsmean[GX][GY], rt, rz, rlon, rlat, robs,
04134         t0, t1, area, dz, dlon, dlat, lon, lat, z, press, temp, rho_air, vmr, h2o,
04135         o3, cw[3];
04136
04137     static int obscount[GX][GY], ip, ix, iy, iz, okay, ci[3];
04138
04139     /* Init... */
04140     if (t == ctl->t_start) {
04141
04142         /* Check quantity index for mass... */
04143         if (ctl->qnt_m < 0)

```

```

04144     ERRMSG("Need quantity mass!");
04145
04146     /* Check dimensions... */
04147     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
04148         ERRMSG("Grid dimensions too large!");
04149
04150     /* Check molar mass... */
04151     if (ctl->molmass <= 0)
04152         ERRMSG("Specify molar mass!");
04153
04154     /* Open observation data file... */
04155     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
04156     if (!(in = fopen(ctl->prof_obsfile, "r")))
04157         ERRMSG("Cannot open file!");
04158
04159     /* Create new output file... */
04160     printf("Write profile data: %s\n", filename);
04161     if (!(out = fopen(filename, "w")))
04162         ERRMSG("Cannot create file!");
04163
04164     /* Write header... */
04165     fprintf(out,
04166         "# $1 = time [s]\n"
04167         "# $2 = altitude [km]\n"
04168         "# $3 = longitude [deg]\n"
04169         "# $4 = latitude [deg]\n"
04170         "# $5 = pressure [hPa]\n"
04171         "# $6 = temperature [K]\n"
04172         "# $7 = volume mixing ratio [ppv]\n"
04173         "# $8 = H2O volume mixing ratio [ppv]\n"
04174         "# $9 = O3 volume mixing ratio [ppv]\n"
04175         "# $10 = observed BT index [K]\n");
04176
04177     /* Set grid box size... */
04178     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
04179     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
04180     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
04181 }
04182
04183 /* Set time interval... */
04184 t0 = t - 0.5 * ctl->dt_mod;
04185 t1 = t + 0.5 * ctl->dt_mod;
04186
04187 /* Initialize... */
04188 #pragma omp parallel for default(shared) private(ix,iy,iz)
04189 for (ix = 0; ix < ctl->prof_nx; ix++)
04190     for (iy = 0; iy < ctl->prof_ny; iy++) {
04191         obsmean[ix][iy] = 0;
04192         obscount[ix][iy] = 0;
04193         for (iz = 0; iz < ctl->prof_nz; iz++)
04194             mass[ix][iy][iz] = 0;
04195     }
04196
04197 /* Read observation data... */
04198 while (fgets(line, LEN, in)) {
04199
04200     /* Read data... */
04201     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rln, &rln, &rln) !=
04202         5)
04203         continue;
04204
04205     /* Check time... */
04206     if (rt < t0)
04207         continue;
04208     if (rt > t1)
04209         break;
04210
04211     /* Calculate indices... */
04212     ix = (int) ((rln - ctl->prof_lon0) / dlon);
04213     iy = (int) ((rln - ctl->prof_lat0) / dlat);
04214
04215     /* Check indices... */
04216     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
04217         continue;
04218
04219     /* Get mean observation index... */
04220     obsmean[ix][iy] += robs;
04221     obscount[ix][iy]++;
04222 }
04223
04224 /* Analyze model data... */
04225 #pragma omp parallel for default(shared) private(ip,ix,iy,iz)
04226 for (ip = 0; ip < atm->np; ip++) {
04227
04228     /* Check time... */
04229     if (atm->time[ip] < t0 || atm->time[ip] > t1)
04230         continue;

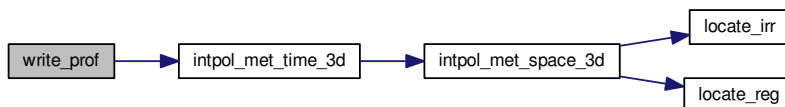
```

```

04231
04232     /* Get indices... */
04233     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
04234     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
04235     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
04236
04237     /* Check indices... */
04238     if (ix < 0 || ix >= ctl->prof_nx ||
04239         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
04240         continue;
04241
04242     /* Get total mass in grid cell... */
04243     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
04244 }
04245
04246 /* Extract profiles... */
04247 for (ix = 0; ix < ctl->prof_nx; ix++)
04248     for (iy = 0; iy < ctl->prof_ny; iy++)
04249         if (obscount[ix][iy] > 0) {
04250
04251             /* Check profile... */
04252             okay = 0;
04253             for (iz = 0; iz < ctl->prof_nz; iz++)
04254                 if (mass[ix][iy][iz] > 0) {
04255                     okay = 1;
04256                     break;
04257                 }
04258             if (!okay)
04259                 continue;
04260
04261             /* Write output... */
04262             fprintf(out, "\n");
04263
04264             /* Loop over altitudes... */
04265             for (iz = 0; iz < ctl->prof_nz; iz++) {
04266
04267                 /* Set coordinates... */
04268                 z = ctl->prof_z0 + dz * (iz + 0.5);
04269                 lon = ctl->prof_lon0 + dlon * (ix + 0.5);
04270                 lat = ctl->prof_lat0 + dlat * (iy + 0.5);
04271
04272                 /* Get pressure and temperature... */
04273                 press = P(z);
04274                 intpol_met_time_3d(met0, met0->t, met1, met1->t, t, press, lon,
04275                                     lat, &temp, ci, cw, 1);
04276                 intpol_met_time_3d(met0, met0->h2o, met1, met1->
04277                                     h2o, t, press, lon,
04278                                     lat, &h2o, ci, cw, 0);
04279                 intpol_met_time_3d(met0, met0->o3, met1, met1->o3, t, press, lon,
04280                                     lat, &o3, ci, cw, 0);
04281
04282                 /* Calculate surface area... */
04283                 area = dlat * dlon * SQR(M_PI * RE / 180.)
04284                     * cos(lat * M_PI / 180.);
04285
04286                 /* Calculate volume mixing ratio... */
04287                 rho_air = 100. * press / (RA * temp);
04288                 vmr = MA / ctl->molmass * mass[ix][iy][iz]
04289                     / (rho_air * area * dz * 1e9);
04290
04291                 /* Write output... */
04292                 fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
04293                         t, z, lon, lat, press, temp, vmr, h2o, o3,
04294                         obsmean[ix][iy] / obscount[ix][iy]);
04295             }
04296         }
04297
04298     /* Close file... */
04299     if (t == ctl->t_stop)
04300         fclose(out);

```

Here is the call graph for this function:



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

Write station data.

Definition at line 4304 of file libtrac.c.

```

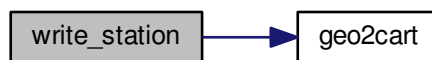
04308     {
04309
04310     static FILE *out;
04311
04312     static double rmax2, t0, t1, x0[3], x1[3];
04313
04314     /* Init... */
04315     if (t == ctl->t_start) {
04316
04317         /* Write info... */
04318         printf("Write station data: %s\n", filename);
04319
04320         /* Create new file... */
04321         if (!(out = fopen(filename, "w")))
04322             ERRMSG("Cannot create file!");
04323
04324         /* Write header... */
04325         fprintf(out,
04326             "# $1 = time [s]\n"
04327             "# $2 = altitude [km]\n"
04328             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
04329         for (int iq = 0; iq < ctl->nq; iq++)
04330             fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
04331                 ctl->qnt_name[iq], ctl->qnt_unit[iq]);
04332         fprintf(out, "\n");
04333
04334         /* Set geolocation and search radius... */
04335         geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
04336         rmax2 = SQR(ctl->stat_r);
04337     }
04338
04339     /* Set time interval for output... */
04340     t0 = t - 0.5 * ctl->dt_mod;
04341     t1 = t + 0.5 * ctl->dt_mod;
04342
04343     /* Loop over air parcels... */
04344     for (int ip = 0; ip < atm->np; ip++) {
04345
04346         /* Check time... */
04347         if (atm->time[ip] < t0 || atm->time[ip] > t1)
04348             continue;
04349
04350         /* Check station flag... */
04351         if (ctl->qnt_stat >= 0)
04352             if (atm->q[ctl->qnt_stat][ip])
04353                 continue;
04354
04355         /* Get Cartesian coordinates... */
04356         geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
04357
04358         /* Check horizontal distance... */
04359         if (DIST2(x0, x1) > rmax2)
04360             continue;
04361
04362         /* Set station flag... */
04363         if (ctl->qnt_stat >= 0)
04364             atm->q[ctl->qnt_stat][ip] = 1;
  
```

```

04365
04366     /* Write data... */
04367     fprintf(out, "%.2f %g %g %g",
04368             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
04369     for (int iq = 0; iq < ctl->nq; iq++) {
04370         fprintf(out, " ");
04371         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
04372     }
04373     fprintf(out, "\n");
04374 }
04375
04376     /* Close file... */
04377     if (t == ctl->t_stop)
04378         fclose(out);
04379 }

```

Here is the call graph for this function:



5.22 libtrac.h

```

00001 /*
00002     This file is part of MPTRAC.
00003
00004     MPTRAC is free software: you can redistribute it and/or modify
00005     it under the terms of the GNU General Public License as published by
00006     the Free Software Foundation, either version 3 of the License, or
00007     (at your option) any later version.
00008
00009     MPTRAC is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU General Public License for more details.
00013
00014     You should have received a copy of the GNU General Public License
00015     along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017     Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00035 #include <ctype.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_spline.h>
00041 #include <gsl/gsl_statistics.h>
00042 #include <math.h>
00043 #include <netcdf.h>
00044 #include <omp.h>
00045 #include <stdio.h>
00046 #include <stdlib.h>
00047 #include <string.h>
00048 #include <time.h>
00049 #include <sys/time.h>
00050
00051 /* -----
00052     Constants...
00053     ----- */
00054
00056 #define G0 9.80665
00057
00059 #define H0 7.0
00060
00062 #define KB 1.3806504e-23
00063

```



```

00065 #define MA 28.9644
00066
00068 #define MH2O 18.01528
00069
00071 #define MO3 48.00
00072
00074 #define P0 1013.25
00075
00077 #define T0 273.15
00078
00080 #define RA 287.058
00081
00083 #define RI 8.3144598
00084
00086 #define RE 6367.421
00087
00088 /* -----
00089     Dimensions...
00090     ----- */
00091
00093 #define LEN 5000
00094
00096 #define NP 10000000
00097
00099 #define NQ 12
00100
00102 #define EP 112
00103
00105 #define EX 1201
00106
00108 #define EY 601
00109
00111 #define GX 720
00112
00114 #define GY 360
00115
00117 #define GZ 100
00118
00120 #define NENS 2000
00121
00123 #define NTHREADS 512
00124
00125 /* -----
00126     Macros...
00127     ----- */
00128
00130 #define ALLOC(ptr, type, n) \
00131     if((ptr=calloc((size_t)(n), sizeof(type)))==NULL) \
00132         ERRMSG("Out of memory!");
00133
00135 #define DEG2DX(dlon, lat) \
00136     ((dlon) * M_PI * RE / 180. * cos((lat) / 180. * M_PI))
00137
00139 #define DEG2DY(dlat) \
00140     ((dlat) * M_PI * RE / 180.)
00141
00143 #define DP2DZ(dp, p) \
00144     (- (dp) * H0 / (p))
00145
00147 #define DX2DEG(dx, lat) \
00148     ((lat) < -89.999 || (lat) > 89.999) ? 0 \
00149     : (dx) * 180. / (M_PI * RE * cos((lat) / 180. * M_PI))
00150
00152 #define DY2DEG(dy) \
00153     ((dy) * 180. / (M_PI * RE))
00154
00156 #define DZ2DP(dz, p) \
00157     (- (dz) * (p) / H0)
00158
00160 #define DIST(a, b) sqrt(DIST2(a, b))
00161
00163 #define DIST2(a, b) \
00164     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00165
00167 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00168
00170 #define ERRMSG(msg) { \
00171     printf("\nError (%s, %s, %d): %s\n\n", \
00172         __FILE__, __func__, __LINE__, msg); \
00173     exit(EXIT_FAILURE); \
00174 }
00175
00177 #define FMOD(x, y) \
00178     ((x) - (int) ((x) / (y)) * (y))
00179
00181 #define FREAD(ptr, type, size, out) { \
00182     if(fread(ptr, sizeof(type), size, out)!=size) \

```

```

00183     ERRMSG("Error while reading!");
00184 }
00185
00187 #define FWRITE(ptr, type, size, out) {
00188     if(fwrite(ptr, sizeof(type), size, out)!=size)
00189     ERRMSG("Error while writing!");
00190 }
00191
00193 #define LIN(x0, y0, x1, y1, x) \
00194     ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00195
00197 #define NC(cmd) {
00198     if((cmd)!=NC_NOERR)
00199     ERRMSG(nc_strerror(cmd));
00200 }
00201
00203 #define NORM(a) sqrt(DOTP(a, a))
00204
00206 #define PRINT(format, var) \
00207     printf("Print (%s, %s, l%d): %s= \"format\"\n",
00208     __FILE__, __func__, __LINE__, #var, var);
00209
00211 #define P(z) (P0 * exp(-(z) / H0))
00212
00214 #define RH(p, t, h2o) (0.263 * 100. * (p) * MH2O / MA * (h2o) \
00215     / exp(17.67 * ((t) - T0) / ((t) - 29.65)))
00216
00218 #define SQ(x) ((x)*(x))
00219
00221 #define THETA(p, t) ((t) * pow(1000. / (p), 0.286))
00222
00224 #define TOK(line, tok, format, var) {
00225     if(((tok)=strtok((line), " \t"))) {
00226         if(sscanf(tok, format, &(var))!=1) continue;
00227     } else ERRMSG("Error while reading!");
00228 }
00229
00231 #define TVIRT(t, h2o) ((t) * (1.0 + 0.609133 * (h2o) * MH2O / MA))
00232
00234 #define WARN(msg) {
00235     printf("\nWarning (%s, %s, l%d): %s\n\n",
00236     __FILE__, __func__, __LINE__, msg);
00237 }
00238
00240 #define Z(p) (H0 * log(P0 / (p)))
00241
00242 /* -----
00243     Timers...
00244     ----- */
00245
00247 #define START_TIMER(id) timer(#id, id, 1)
00248
00250 #define STOP_TIMER(id) timer(#id, id, 2)
00251
00253 #define PRINT_TIMER(id) timer(#id, id, 3)
00254
00256 #define NTIMER 20
00257
00259 #define TIMER_ZERO 0
00260
00262 #define TIMER_INIT 1
00263
00265 #define TIMER_INPUT 2
00266
00268 #define TIMER_OUTPUT 3
00269
00271 #define TIMER_ADVECT 4
00272
00274 #define TIMER_DECAY 5
00275
00277 #define TIMER_DIFFMESO 6
00278
00280 #define TIMER_DIFFTURB 7
00281
00283 #define TIMER_ISOSURF 8
00284
00286 #define TIMER_METEO 9
00287
00289 #define TIMER_POSITION 10
00290
00292 #define TIMER_SEDI 11
00293
00295 #define TIMER_OHCHEM 12
00296
00298 #define TIMER_WETDEPO 13
00299
00301 #define TIMER_TOTAL 14

```

```
00302
00303 /* -----
00304    Structs...
00305    ----- */
00306
00308 typedef struct {
00309     int nq;
00312
00314     char qnt_name[NQ] [LEN];
00315
00317     char qnt_unit[NQ] [LEN];
00318
00320     char qnt_format[NQ] [LEN];
00321
00323     int qnt_ens;
00324
00326     int qnt_m;
00327
00329     int qnt_rho;
00330
00332     int qnt_r;
00333
00335     int qnt_ps;
00336
00338     int qnt_pt;
00339
00341     int qnt_z;
00342
00344     int qnt_p;
00345
00347     int qnt_t;
00348
00350     int qnt_u;
00351
00353     int qnt_v;
00354
00356     int qnt_w;
00357
00359     int qnt_h2o;
00360
00362     int qnt_o3;
00363
00365     int qnt_lwc;
00366
00368     int qnt_iwc;
00369
00371     int qnt_pc;
00372
00374     int qnt_hno3;
00375
00377     int qnt_oh;
00378
00380     int qnt_rh;
00381
00383     int qnt_theta;
00384
00386     int qnt_vh;
00387
00389     int qnt_vz;
00390
00392     int qnt_pv;
00393
00395     int qnt_tice;
00396
00398     int qnt_tsts;
00399
00401     int qnt_tnat;
00402
00404     int qnt_stat;
00405
00407     int direction;
00408
00410     double t_start;
00411
00413     double t_stop;
00414
00416     double dt_mod;
00417
00419     double dt_met;
00420
00422     int met_dx;
00423
00425     int met_dy;
00426
00428     int met_dp;
00429
```

```
00431  int met_sx;
00432
00434  int met_sy;
00435
00437  int met_sp;
00438
00440  int met_np;
00441
00443  double met_p[EP];
00444
00447  int met_tropo;
00448
00450  char met_geopot[LEN];
00451
00453  double met_dt_out;
00454
00456  char met_stage[LEN];
00457
00460  int isosurf;
00461
00463  char balloon[LEN];
00464
00466  double turb_dx_trop;
00467
00469  double turb_dx_strat;
00470
00472  double turb_dz_trop;
00473
00475  double turb_dz_strat;
00476
00478  double turb_mesox;
00479
00481  double turb_mesoz;
00482
00484  char species[LEN];
00485
00487  double molmass;
00488
00490  double tdec_trop;
00491
00493  double tdec_strat;
00494
00496  double oh_chem[4];
00497
00499  double wet_depo[4];
00500
00502  double psc_h2o;
00503
00505  double psc_hno3;
00506
00508  char atm_basename[LEN];
00509
00511  char atm_gpfile[LEN];
00512
00514  double atm_dt_out;
00515
00517  int atm_filter;
00518
00520  int atm_stride;
00521
00523  int atm_type;
00524
00526  char csi_basename[LEN];
00527
00529  double csi_dt_out;
00530
00532  char csi_obsfile[LEN];
00533
00535  double csi_obsmin;
00536
00538  double csi_modmin;
00539
00541  int csi_nz;
00542
00544  double csi_z0;
00545
00547  double csi_z1;
00548
00550  int csi_nx;
00551
00553  double csi_lon0;
00554
00556  double csi_lon1;
00557
00559  int csi_ny;
00560
00562  double csi_lat0;
```

```
00563
00565 double csi_lat1;
00566
00568 char grid_basename[LEN];
00569
00571 char grid_gpfile[LEN];
00572
00574 double grid_dt_out;
00575
00577 int grid_sparse;
00578
00580 int grid_nz;
00581
00583 double grid_z0;
00584
00586 double grid_z1;
00587
00589 int grid_nx;
00590
00592 double grid_lon0;
00593
00595 double grid_lon1;
00596
00598 int grid_ny;
00599
00601 double grid_lat0;
00602
00604 double grid_lat1;
00605
00607 char prof_basename[LEN];
00608
00610 char prof_obsfile[LEN];
00611
00613 int prof_nz;
00614
00616 double prof_z0;
00617
00619 double prof_z1;
00620
00622 int prof_nx;
00623
00625 double prof_lon0;
00626
00628 double prof_lon1;
00629
00631 int prof_ny;
00632
00634 double prof_lat0;
00635
00637 double prof_lat1;
00638
00640 char ens_basename[LEN];
00641
00643 char stat_basename[LEN];
00644
00646 double stat_lon;
00647
00649 double stat_lat;
00650
00652 double stat_r;
00653
00654 } ctl_t;
00655
00657 typedef struct {
00658
00660 int np;
00661
00663 double time[NP];
00664
00666 double p[NP];
00667
00669 double lon[NP];
00670
00672 double lat[NP];
00673
00675 double q[NQ][NP];
00676
00677 } atm_t;
00678
00680 typedef struct {
00681
00683 float up[NP];
00684
00686 float vp[NP];
00687
00689 float wp[NP];
00690
```

```

00692 double iso_var[NP];
00693
00695 double iso_ps[NP];
00696
00698 double iso_ts[NP];
00699
00701 int iso_n;
00702
00704 double tsig[EX][EY][EP];
00705
00707 float usig[EX][EY][EP];
00708
00710 float vsig[EX][EY][EP];
00711
00713 float wsig[EX][EY][EP];
00714
00715 } cache_t;
00716
00718 typedef struct {
00719
00721 double time;
00722
00724 int nx;
00725
00727 int ny;
00728
00730 int np;
00731
00733 double lon[EX];
00734
00736 double lat[EY];
00737
00739 double p[EP];
00740
00742 float ps[EX][EY];
00743
00745 float zs[EX][EY];
00746
00748 float pt[EX][EY];
00749
00751 float pc[EX][EY];
00752
00754 float cl[EX][EY];
00755
00757 float z[EX][EY][EP];
00758
00760 float t[EX][EY][EP];
00761
00763 float u[EX][EY][EP];
00764
00766 float v[EX][EY][EP];
00767
00769 float w[EX][EY][EP];
00770
00772 float pv[EX][EY][EP];
00773
00775 float h2o[EX][EY][EP];
00776
00778 float o3[EX][EY][EP];
00779
00781 float lwc[EX][EY][EP];
00782
00784 float iwc[EX][EY][EP];
00785
00787 float pl[EX][EY][EP];
00788
00789 } met_t;
00790
00791 /* -----
00792 Functions...
00793 ----- */
00794
00796 void cart2geo(
00797 double *x,
00798 double *z,
00799 double *lon,
00800 double *lat);
00801
00803 #ifdef _OPENACC
00804 #pragma acc routine (check_finite)
00805 #endif
00806 int check_finite(
00807 const double x);
00808
00810 #ifdef _OPENACC
00811 #pragma acc routine (clim_hno3)
00812 #endif

```

```
00813 double clim_hno3(
00814     double t,
00815     double lat,
00816     double p);
00817
00819 #ifdef _OPENACC
00820 #pragma acc routine (clim_oh)
00821 #endif
00822 double clim_oh(
00823     double t,
00824     double lat,
00825     double p);
00826
00828 #ifdef _OPENACC
00829 #pragma acc routine (clim_tropo)
00830 #endif
00831 double clim_tropo(
00832     double t,
00833     double lat);
00834
00836 void day2doy(
00837     int year,
00838     int mon,
00839     int day,
00840     int *doy);
00841
00843 void doy2day(
00844     int year,
00845     int doy,
00846     int *mon,
00847     int *day);
00848
00850 void geo2cart(
00851     double z,
00852     double lon,
00853     double lat,
00854     double *x);
00855
00857 void get_met(
00858     ctl_t * ctl,
00859     char *metbase,
00860     double t,
00861     met_t ** met0,
00862     met_t ** met1);
00863
00865 void get_met_help(
00866     double t,
00867     int direct,
00868     char *metbase,
00869     double dt_met,
00870     char *filename);
00871
00873 void get_met_replace(
00874     char *orig,
00875     char *search,
00876     char *repl);
00877
00879 #ifdef _OPENACC
00880 #pragma acc routine (intpol_met_space_3d)
00881 #endif
00882 void intpol_met_space_3d(
00883     met_t * met,
00884     float array[EX][EY][EP],
00885     double p,
00886     double lon,
00887     double lat,
00888     double *var,
00889     int *ci,
00890     double *cw,
00891     int init);
00892
00894 #ifdef _OPENACC
00895 #pragma acc routine (intpol_met_space_2d)
00896 #endif
00897 void intpol_met_space_2d(
00898     met_t * met,
00899     float array[EX][EY],
00900     double lon,
00901     double lat,
00902     double *var,
00903     int *ci,
00904     double *cw,
00905     int init);
00906
00908 #ifdef _OPENACC
00909 #pragma acc routine (intpol_met_time_3d)
00910 #endif
```

```

00911 void intpol_met_time_3d(
00912     met_t * met0,
00913     float array0[EX][EY][EP],
00914     met_t * met1,
00915     float array1[EX][EY][EP],
00916     double ts,
00917     double p,
00918     double lon,
00919     double lat,
00920     double *var,
00921     int *ci,
00922     double *cw,
00923     int init);
00924
00926 #ifdef _OPENACC
00927 #pragma acc routine (intpol_met_time_2d)
00928 #endif
00929 void intpol_met_time_2d(
00930     met_t * met0,
00931     float array0[EX][EY],
00932     met_t * met1,
00933     float array1[EX][EY],
00934     double ts,
00935     double lon,
00936     double lat,
00937     double *var,
00938     int *ci,
00939     double *cw,
00940     int init);
00941
00943 void jsec2time(
00944     double jsec,
00945     int *year,
00946     int *mon,
00947     int *day,
00948     int *hour,
00949     int *min,
00950     int *sec,
00951     double *remain);
00952
00954 #ifdef _OPENACC
00955 #pragma acc routine (locate_irr)
00956 #endif
00957 int locate_irr(
00958     double **xx,
00959     int n,
00960     double x);
00961
00963 #ifdef _OPENACC
00964 #pragma acc routine (locate_reg)
00965 #endif
00966 int locate_reg(
00967     double **xx,
00968     int n,
00969     double x);
00970
00972 int read_atm(
00973     const char *filename,
00974     ctl_t * ctl,
00975     atm_t * atm);
00976
00978 void read_ctl(
00979     const char *filename,
00980     int argc,
00981     char *argv[],
00982     ctl_t * ctl);
00983
00985 int read_met(
00986     ctl_t * ctl,
00987     char *filename,
00988     met_t * met);
00989
00991 void read_met_cloud(
00992     met_t * met);
00993
00995 void read_met_extrapolate(
00996     met_t * met);
00997
00999 void read_met_geopot(
01000     met_t * met);
01001
01003 int read_met_help_3d(
01004     int ncid,
01005     char *varname,
01006     char *varname2,
01007     met_t * met,
01008     float dest[EX][EY][EP],

```



```
01009     float scl);
01010
01012 int read_met_help_2d(
01013     int ncid,
01014     char *varname,
01015     char *varname2,
01016     met_t * met,
01017     float dest[EX][EY],
01018     float scl);
01019
01021 void read_met_ml2pl(
01022     ctl_t * ctl,
01023     met_t * met,
01024     float var[EX][EY][EP]);
01025
01027 void read_met_periodic(
01028     met_t * met);
01029
01031 void read_met_pv(
01032     met_t * met);
01033
01035 void read_met_sample(
01036     ctl_t * ctl,
01037     met_t * met);
01038
01040 void read_met_surface(
01041     int ncid,
01042     met_t * met);
01043
01045 void read_met_tropo(
01046     ctl_t * ctl,
01047     met_t * met);
01048
01050 double scan_ctl(
01051     const char *filename,
01052     int argc,
01053     char *argv[],
01054     const char *varname,
01055     int arridx,
01056     const char *defvalue,
01057     char *value);
01058
01060 void spline(
01061     double *x,
01062     double *y,
01063     int n,
01064     double *x2,
01065     double *y2,
01066     int n2);
01067
01069 #ifdef _OPENACC
01070 #pragma acc routine (stddev)
01071 #endif
01072 double stddev(
01073     double *data,
01074     int n);
01075
01077 void time2jsec(
01078     int year,
01079     int mon,
01080     int day,
01081     int hour,
01082     int min,
01083     int sec,
01084     double remain,
01085     double *jsec);
01086
01088 void timer(
01089     const char *name,
01090     int id,
01091     int mode);
01092
01094 void write_atm(
01095     const char *filename,
01096     ctl_t * ctl,
01097     atm_t * atm,
01098     double t);
01099
01101 void write_csi(
01102     const char *filename,
01103     ctl_t * ctl,
01104     atm_t * atm,
01105     double t);
01106
01108 void write_ens(
01109     const char *filename,
01110     ctl_t * ctl,
```

```

01111     atm_t * atm,
01112     double t);
01113
01115 void write_grid(
01116     const char *filename,
01117     ctl_t * ctl,
01118     met_t * met0,
01119     met_t * met1,
01120     atm_t * atm,
01121     double t);
01122
01124 void write_prof(
01125     const char *filename,
01126     ctl_t * ctl,
01127     met_t * met0,
01128     met_t * met1,
01129     atm_t * atm,
01130     double t);
01131
01133 void write_station(
01134     const char *filename,
01135     ctl_t * ctl,
01136     atm_t * atm,
01137     double t);

```

5.23 met_map.c File Reference

Extract map from meteorological data.

Functions

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

5.23.1 Detailed Description

Extract map from meteorological data.

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

5.23.2 Function Documentation

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

Definition at line 41 of file [met_map.c](#).

```

00043         {
00044
00045         ctl_t ctl;
00046
00047         met_t *met;
00048
00049         FILE *out;
00050
00051         static double timem[NX][NY], p0, ps, psm[NX][NY], pt, ptm[NX][NY], t,
00052         tm[NX][NY], u, um[NX][NY], v, vm[NX][NY], w, wm[NX][NY], h2o,
00053         h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00054         lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
00055         pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY],
00056         pc, pcm[NX][NY], cl, clm[NX][NY], lon, lon0, lon1, lons[NX],
00057         dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00058
00059         static int i, ix, iy, np[NX][NY], nx, ny, ci[3];
00060
00061         /* Allocate... */
00062         ALLOC(met, met_t, 1);

```

```

00063
00064 /* Check arguments... */
00065 if (argc < 4)
00066     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00067
00068 /* Read control parameters... */
00069 read_ctl(argv[1], argc, argv, &ctl);
00070 p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00071 lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00072 lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00073 dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00074 lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00075 lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00076 dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00077
00078 /* Loop over files... */
00079 for (i = 3; i < argc; i++) {
00080
00081     /* Read meteorological data... */
00082     if (!read_met(&ctl, argv[i], met))
00083         continue;
00084
00085     /* Set horizontal grid... */
00086     if (dlon <= 0)
00087         dlon = fabs(met->lon[1] - met->lon[0]);
00088     if (dlat <= 0)
00089         dlat = fabs(met->lat[1] - met->lat[0]);
00090     if (lon0 < -360 && lon1 > 360) {
00091         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00092         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00093     }
00094     nx = ny = 0;
00095     for (lon = lon0; lon <= lon1; lon += dlon) {
00096         lons[nx] = lon;
00097         if ((++nx) > NX)
00098             ERRMSG("Too many longitudes!");
00099     }
00100     if (lat0 < -90 && lat1 > 90) {
00101         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00102         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00103     }
00104     for (lat = lat0; lat <= lat1; lat += dlat) {
00105         lats[ny] = lat;
00106         if ((++ny) > NY)
00107             ERRMSG("Too many latitudes!");
00108     }
00109
00110     /* Average... */
00111     for (ix = 0; ix < nx; ix++)
00112         for (iy = 0; iy < ny; iy++) {
00113
00114             /* Interpolate meteo data... */
00115             intpol_met_space_3d(met, met->z, p0, lons[ix], lats[iy], &z, ci, cw,
00116                                 1);
00117             intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy], &t, ci, cw,
00118                                 0);
00119             intpol_met_space_3d(met, met->u, p0, lons[ix], lats[iy], &u, ci, cw,
00120                                 0);
00121             intpol_met_space_3d(met, met->v, p0, lons[ix], lats[iy], &v, ci, cw,
00122                                 0);
00123             intpol_met_space_3d(met, met->w, p0, lons[ix], lats[iy], &w, ci, cw,
00124                                 0);
00125             intpol_met_space_3d(met, met->pv, p0, lons[ix], lats[iy], &pv, ci,
00126                                 cw, 0);
00127             intpol_met_space_3d(met, met->h2o, p0, lons[ix], lats[iy], &h2o, ci,
00128                                 cw, 0);
00129             intpol_met_space_3d(met, met->o3, p0, lons[ix], lats[iy], &o3, ci,
00130                                 cw, 0);
00131             intpol_met_space_3d(met, met->lwc, p0, lons[ix], lats[iy], &lwc, ci,
00132                                 cw, 0);
00133             intpol_met_space_3d(met, met->iwc, p0, lons[ix], lats[iy], &iwc, ci,
00134                                 cw, 0);
00135             intpol_met_space_2d(met, met->ps, lons[ix], lats[iy], &ps, ci, cw, 0);
00136             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy], &pt, ci, cw, 0);
00137             intpol_met_space_2d(met, met->pc, lons[ix], lats[iy], &pc, ci, cw, 0);
00138             intpol_met_space_2d(met, met->c1, lons[ix], lats[iy], &c1, ci, cw, 0);
00139
00140             /* Interpolate tropopause data... */
00141             intpol_met_space_3d(met, met->z, pt, lons[ix], lats[iy], &zt, ci, cw,
00142                                 1);
00143             intpol_met_space_3d(met, met->t, pt, lons[ix], lats[iy], &tt, ci, cw,
00144                                 0);
00145             intpol_met_space_3d(met, met->h2o, pt, lons[ix], lats[iy], &h2ot, ci,
00146                                 cw, 0);
00147
00148             /* Averaging... */
00149             timem[ix][iy] += met->time;

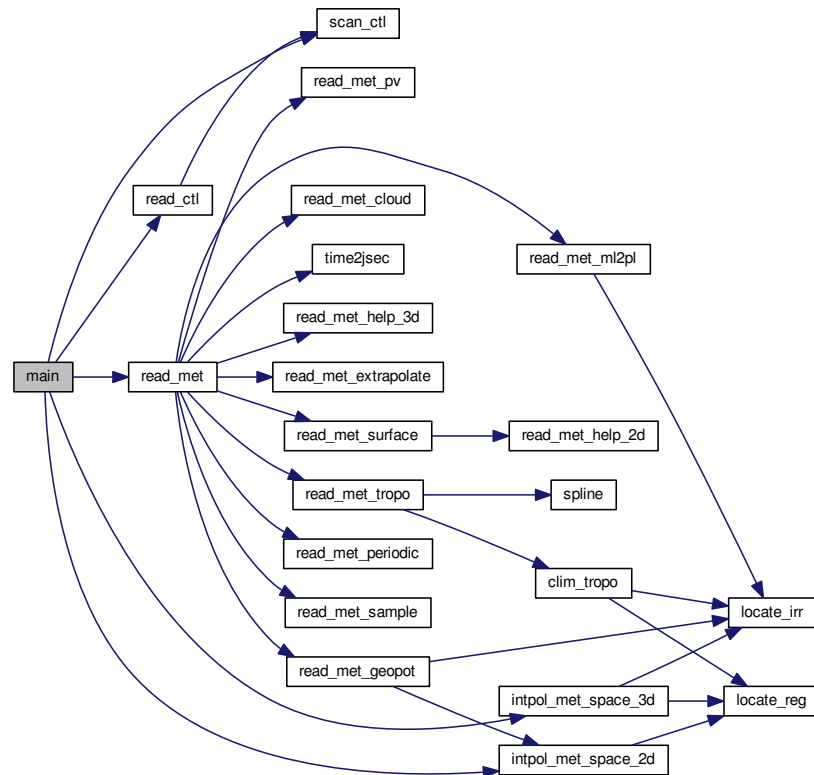
```

```

00150         zm[ix][iy] += z;
00151         tm[ix][iy] += t;
00152         um[ix][iy] += u;
00153         vm[ix][iy] += v;
00154         wm[ix][iy] += w;
00155         pvm[ix][iy] += pv;
00156         h2om[ix][iy] += h2o;
00157         o3m[ix][iy] += o3;
00158         lwcm[ix][iy] += lwc;
00159         iwcm[ix][iy] += iwc;
00160         psm[ix][iy] += ps;
00161         ptm[ix][iy] += pt;
00162         pcm[ix][iy] += pc;
00163         clm[ix][iy] += cl;
00164         ztm[ix][iy] += zt;
00165         ttm[ix][iy] += tt;
00166         h2otm[ix][iy] += h2ot;
00167         np[ix][iy]++;
00168     }
00169 }
00170
00171 /* Create output file... */
00172 printf("Write meteorological data file: %s\n", argv[2]);
00173 if (!(out = fopen(argv[2], "w")))
00174     ERRMSG("Cannot create file!");
00175
00176 /* Write header... */
00177 fprintf(out,
00178         "# $1 = time [s]\n"
00179         "# $2 = altitude [km]\n"
00180         "# $3 = longitude [deg]\n"
00181         "# $4 = latitude [deg]\n"
00182         "# $5 = pressure [hPa]\n"
00183         "# $6 = temperature [K]\n"
00184         "# $7 = zonal wind [m/s]\n"
00185         "# $8 = meridional wind [m/s]\n"
00186         "# $9 = vertical wind [hPa/s]\n"
00187         "# $10 = H2O volume mixing ratio [ppv]\n");
00188 fprintf(out,
00189         "# $11 = O3 volume mixing ratio [ppv]\n"
00190         "# $12 = geopotential height [km]\n"
00191         "# $13 = potential vorticity [PVU]\n"
00192         "# $14 = surface pressure [hPa]\n"
00193         "# $15 = tropopause pressure [hPa]\n"
00194         "# $16 = tropopause geopotential height [km]\n"
00195         "# $17 = tropopause temperature [K]\n"
00196         "# $18 = tropopause water vapor [ppv]\n"
00197         "# $19 = cloud liquid water content [kg/kg]\n"
00198         "# $20 = cloud ice water content [kg/kg]\n");
00199 fprintf(out,
00200         "# $21 = total column cloud water [kg/m^2]\n"
00201         "# $22 = cloud top pressure [hPa]\n");
00202
00203 /* Write data... */
00204 for (iy = 0; iy < ny; iy++) {
00205     fprintf(out, "\n");
00206     for (ix = 0; ix < nx; ix++)
00207         fprintf(out,
00208                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00209                 timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
00210                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00211                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00212                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00213                 zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00214                 psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00215                 ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy],
00216                 h2otm[ix][iy] / np[ix][iy], lwcm[ix][iy] / np[ix][iy],
00217                 iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00218                 pcm[ix][iy] / np[ix][iy]);
00219 }
00220
00221 /* Close file... */
00222 fclose(out);
00223
00224 /* Free... */
00225 free(met);
00226
00227 return EXIT_SUCCESS;
00228 }

```

Here is the call graph for this function:



5.24 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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00020 #include "libtrac.h"
00021
00022 /* -----
00023  Dimensions...
00024  ----- */
00025
00026 #define NX 1441
00027
00028 #define NY 721
00029
00030 /* -----
00031  Main...
00032  ----- */
00033
00034 int main(

```

```

00042     int argc,
00043     char *argv[] {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NX][NY], p0, ps, psm[NX][NY], pt, ptm[NX][NY], t,
00052     tm[NX][NY], u, um[NX][NY], v, vm[NX][NY], w, wm[NX][NY], h2o,
00053     h2om[NX][NY], h2ot, h2otm[NX][NY], o3, o3m[NX][NY],
00054     lwc, lwcm[NX][NY], iwc, iwcm[NX][NY], z, zm[NX][NY], pv,
00055     pvm[NX][NY], zt, ztm[NX][NY], tt, ttm[NX][NY],
00056     pc, pcm[NX][NY], cl, clm[NX][NY], lon, lon0, lon1, lons[NX],
00057     dlon, lat, lat0, lat1, lats[NY], dlat, cw[3];
00058
00059     static int i, ix, iy, np[NX][NY], nx, ny, ci[3];
00060
00061     /* Allocate... */
00062     ALLOC(met, met_t, 1);
00063
00064     /* Check arguments... */
00065     if (argc < 4)
00066         ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00067
00068     /* Read control parameters... */
00069     read_ctl(argv[1], argc, argv, &ctl);
00070     p0 = P(scan_ctl(argv[1], argc, argv, "MAP_Z0", -1, "10", NULL));
00071     lon0 = scan_ctl(argv[1], argc, argv, "MAP_LON0", -1, "-180", NULL);
00072     lon1 = scan_ctl(argv[1], argc, argv, "MAP_LON1", -1, "180", NULL);
00073     dlon = scan_ctl(argv[1], argc, argv, "MAP_DLON", -1, "-999", NULL);
00074     lat0 = scan_ctl(argv[1], argc, argv, "MAP_LAT0", -1, "-90", NULL);
00075     lat1 = scan_ctl(argv[1], argc, argv, "MAP_LAT1", -1, "90", NULL);
00076     dlat = scan_ctl(argv[1], argc, argv, "MAP_DLAT", -1, "-999", NULL);
00077
00078     /* Loop over files... */
00079     for (i = 3; i < argc; i++) {
00080
00081         /* Read meteorological data... */
00082         if (!read_met(&ctl, argv[i], met))
00083             continue;
00084
00085         /* Set horizontal grid... */
00086         if (dlon <= 0)
00087             dlon = fabs(met->lon[1] - met->lon[0]);
00088         if (dlat <= 0)
00089             dlat = fabs(met->lat[1] - met->lat[0]);
00090         if (lon0 < -360 && lon1 > 360) {
00091             lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00092             lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00093         }
00094         nx = ny = 0;
00095         for (lon = lon0; lon <= lon1; lon += dlon) {
00096             lons[nx] = lon;
00097             if ((++nx) > NX)
00098                 ERRMSG("Too many longitudes!");
00099         }
00100         if (lat0 < -90 && lat1 > 90) {
00101             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00102             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00103         }
00104         for (lat = lat0; lat <= lat1; lat += dlat) {
00105             lats[ny] = lat;
00106             if ((++ny) > NY)
00107                 ERRMSG("Too many latitudes!");
00108         }
00109
00110         /* Average... */
00111         for (ix = 0; ix < nx; ix++)
00112             for (iy = 0; iy < ny; iy++) {
00113
00114                 /* Interpolate meteo data... */
00115                 intpol_met_space_3d(met, met->z, p0, lons[ix], lats[iy], &z, ci, cw,
00116                                     1);
00117                 intpol_met_space_3d(met, met->t, p0, lons[ix], lats[iy], &t, ci, cw,
00118                                     0);
00119                 intpol_met_space_3d(met, met->u, p0, lons[ix], lats[iy], &u, ci, cw,
00120                                     0);
00121                 intpol_met_space_3d(met, met->v, p0, lons[ix], lats[iy], &v, ci, cw,
00122                                     0);
00123                 intpol_met_space_3d(met, met->w, p0, lons[ix], lats[iy], &w, ci, cw,
00124                                     0);
00125                 intpol_met_space_3d(met, met->pv, p0, lons[ix], lats[iy], &pv, ci,
00126                                     cw, 0);
00127                 intpol_met_space_3d(met, met->h2o, p0, lons[ix], lats[iy], &h2o, ci,
00128                                     cw, 0);

```

```

00129     intpol_met_space_3d(met, met->o3, p0, lons[ix], lats[iy], &o3, ci,
00130                          cw, 0);
00131     intpol_met_space_3d(met, met->lwc, p0, lons[ix], lats[iy], &lwc, ci,
00132                          cw, 0);
00133     intpol_met_space_3d(met, met->iwc, p0, lons[ix], lats[iy], &iwc, ci,
00134                          cw, 0);
00135     intpol_met_space_2d(met, met->ps, lons[ix], lats[iy], &ps, ci, cw, 0);
00136     intpol_met_space_2d(met, met->pt, lons[ix], lats[iy], &pt, ci, cw, 0);
00137     intpol_met_space_2d(met, met->pc, lons[ix], lats[iy], &pc, ci, cw, 0);
00138     intpol_met_space_2d(met, met->cl, lons[ix], lats[iy], &cl, ci, cw, 0);
00139
00140     /* Interpolate tropopause data... */
00141     intpol_met_space_3d(met, met->z, pt, lons[ix], lats[iy], &zt, ci, cw,
00142                          1);
00143     intpol_met_space_3d(met, met->t, pt, lons[ix], lats[iy], &tt, ci, cw,
00144                          0);
00145     intpol_met_space_3d(met, met->h2o, pt, lons[ix], lats[iy], &h2ot, ci,
00146                          cw, 0);
00147
00148     /* Averaging... */
00149     timem[ix][iy] += met->time;
00150     zm[ix][iy] += z;
00151     tm[ix][iy] += t;
00152     um[ix][iy] += u;
00153     vm[ix][iy] += v;
00154     wm[ix][iy] += w;
00155     pvm[ix][iy] += pv;
00156     h2om[ix][iy] += h2o;
00157     o3m[ix][iy] += o3;
00158     lwcm[ix][iy] += lwc;
00159     iwcm[ix][iy] += iwc;
00160     psm[ix][iy] += ps;
00161     ptm[ix][iy] += pt;
00162     pcm[ix][iy] += pc;
00163     clm[ix][iy] += cl;
00164     ztm[ix][iy] += zt;
00165     ttm[ix][iy] += tt;
00166     h2otm[ix][iy] += h2ot;
00167     np[ix][iy]++;
00168 }
00169 }
00170
00171 /* Create output file... */
00172 printf("Write meteorological data file: %s\n", argv[2]);
00173 if (!(out = fopen(argv[2], "w")))
00174     ERRMSG("Cannot create file!");
00175
00176 /* Write header... */
00177 fprintf(out,
00178         "# $1 = time [s]\n"
00179         "# $2 = altitude [km]\n"
00180         "# $3 = longitude [deg]\n"
00181         "# $4 = latitude [deg]\n"
00182         "# $5 = pressure [hPa]\n"
00183         "# $6 = temperature [K]\n"
00184         "# $7 = zonal wind [m/s]\n"
00185         "# $8 = meridional wind [m/s]\n"
00186         "# $9 = vertical wind [hPa/s]\n"
00187         "# $10 = H2O volume mixing ratio [ppv]\n");
00188 fprintf(out,
00189         "# $11 = O3 volume mixing ratio [ppv]\n"
00190         "# $12 = geopotential height [km]\n"
00191         "# $13 = potential vorticity [PVU]\n"
00192         "# $14 = surface pressure [hPa]\n"
00193         "# $15 = tropopause pressure [hPa]\n"
00194         "# $16 = tropopause geopotential height [km]\n"
00195         "# $17 = tropopause temperature [K]\n"
00196         "# $18 = tropopause water vapor [ppv]\n"
00197         "# $19 = cloud liquid water content [kg/kg]\n"
00198         "# $20 = cloud ice water content [kg/kg]\n");
00199 fprintf(out,
00200         "# $21 = total column cloud water [kg/m^2]\n"
00201         "# $22 = cloud top pressure [hPa]\n");
00202
00203 /* Write data... */
00204 for (iy = 0; iy < ny; iy++) {
00205     fprintf(out, "\n");
00206     for (ix = 0; ix < nx; ix++)
00207         fprintf(out,
00208                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00209                 timem[ix][iy] / np[ix][iy], Z(p0), lons[ix], lats[iy], p0,
00210                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00211                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00212                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00213                 zm[ix][iy] / np[ix][iy], pvm[ix][iy] / np[ix][iy],
00214                 psm[ix][iy] / np[ix][iy], ptm[ix][iy] / np[ix][iy],
00215                 ztm[ix][iy] / np[ix][iy], ttm[ix][iy] / np[ix][iy],

```

```

00216             h2otm[ix][iy] / np[ix][iy], lwcm[ix][iy] / np[ix][iy],
00217             iwcm[ix][iy] / np[ix][iy], clm[ix][iy] / np[ix][iy],
00218             pcm[ix][iy] / np[ix][iy]);
00219     }
00220
00221     /* Close file... */
00222     fclose(out);
00223
00224     /* Free... */
00225     free(met);
00226
00227     return EXIT_SUCCESS;
00228 }

```

5.25 met_prof.c File Reference

Extract vertical profile from meteorological data.

Functions

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

5.25.1 Detailed Description

Extract vertical profile from meteorological data.

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

5.25.2 Function Documentation

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

Definition at line 38 of file [met_prof.c](#).

```

00040     {
00041
00042     ctl_t ctl;
00043
00044     met_t *met;
00045
00046     FILE *out;
00047
00048     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050     wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00051     iwc, iwcm[NZ], ps, psm[NZ], pt, ptm[NZ], pc, pcm[NZ], cl, clm[NZ],
00052     tt, ttm[NZ], zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ], cw[3];
00053
00054     static int i, iz, np[NZ], npt[NZ], nz, ci[3];
00055
00056     /* Allocate... */
00057     ALLOC(met, met_t, 1);
00058
00059     /* Check arguments... */
00060     if (argc < 4)
00061         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00066     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00067     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00068     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00069     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00070     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);

```



```

00071 lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00072 lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00073 dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00074
00075 /* Loop over input files... */
00076 for (i = 3; i < argc; i++) {
00077
00078     /* Read meteorological data... */
00079     if (!read_met(&ctl, argv[i], met))
00080         continue;
00081
00082     /* Set vertical grid... */
00083     if (z0 < 0)
00084         z0 = Z(met->p[0]);
00085     if (z1 < 0)
00086         z1 = Z(met->p[met->np - 1]);
00087     nz = 0;
00088     if (dz < 0) {
00089         for (iz = 0; iz < met->np; iz++)
00090             if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00091                 plev[nz] = met->p[iz];
00092                 if ((++nz) > NZ)
00093                     ERRMSG("Too many pressure levels!");
00094             }
00095     } else
00096         for (z = z0; z <= z1; z += dz) {
00097             plev[nz] = P(z);
00098             if ((++nz) > NZ)
00099                 ERRMSG("Too many pressure levels!");
00100         }
00101
00102     /* Set horizontal grid... */
00103     if (dlon <= 0)
00104         dlon = fabs(met->lon[1] - met->lon[0]);
00105     if (dlat <= 0)
00106         dlat = fabs(met->lat[1] - met->lat[0]);
00107
00108     /* Average... */
00109     for (iz = 0; iz < nz; iz++)
00110         for (lon = lon0; lon <= lon1; lon += dlon)
00111             for (lat = lat0; lat <= lat1; lat += dlat) {
00112
00113                 /* Interpolate meteo data... */
00114                 intpol_met_space_3d(met, met->z, plev[iz], lon, lat, &z, ci, cw, 1);
00115                 intpol_met_space_3d(met, met->t, plev[iz], lon, lat, &t, ci, cw, 0);
00116                 intpol_met_space_3d(met, met->u, plev[iz], lon, lat, &u, ci, cw, 0);
00117                 intpol_met_space_3d(met, met->v, plev[iz], lon, lat, &v, ci, cw, 0);
00118                 intpol_met_space_3d(met, met->w, plev[iz], lon, lat, &w, ci, cw, 0);
00119                 intpol_met_space_3d(met, met->pv, plev[iz], lon, lat, &pv, ci, cw,
00120                                     0);
00121                 intpol_met_space_3d(met, met->h2o, plev[iz], lon, lat, &h2o, ci, cw,
00122                                     0);
00123                 intpol_met_space_3d(met, met->o3, plev[iz], lon, lat, &o3, ci, cw,
00124                                     0);
00125                 intpol_met_space_3d(met, met->lwc, plev[iz], lon, lat, &lwc, ci, cw,
00126                                     0);
00127                 intpol_met_space_3d(met, met->iwc, plev[iz], lon, lat, &iwc, ci, cw,
00128                                     0);
00129                 intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
00130                 intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
00131                 intpol_met_space_2d(met, met->pc, lon, lat, &pc, ci, cw, 0);
00132                 intpol_met_space_2d(met, met->cl, lon, lat, &cl, ci, cw, 0);
00133
00134                 /* Interpolate tropopause data... */
00135                 intpol_met_space_3d(met, met->z, pt, lon, lat, &zt, ci, cw, 1);
00136                 intpol_met_space_3d(met, met->t, pt, lon, lat, &tt, ci, cw, 0);
00137                 intpol_met_space_3d(met, met->h2o, pt, lon, lat, &h2ot, ci, cw, 0);
00138
00139                 /* Averaging... */
00140                 if (gsl_finite(t) && gsl_finite(u)
00141                     && gsl_finite(v) && gsl_finite(w)) {
00142                     timem[iz] += met->time;
00143                     lonm[iz] += lon;
00144                     latm[iz] += lat;
00145                     zm[iz] += z;
00146                     tm[iz] += t;
00147                     um[iz] += u;
00148                     vm[iz] += v;
00149                     wm[iz] += w;
00150                     pvm[iz] += pv;
00151                     h2om[iz] += h2o;
00152                     o3m[iz] += o3;
00153                     psm[iz] += ps;
00154                     pcm[iz] += pc;
00155                     clm[iz] += cl;
00156                     lwcm[iz] += lwc;
00157                     iwcm[iz] += iwc;

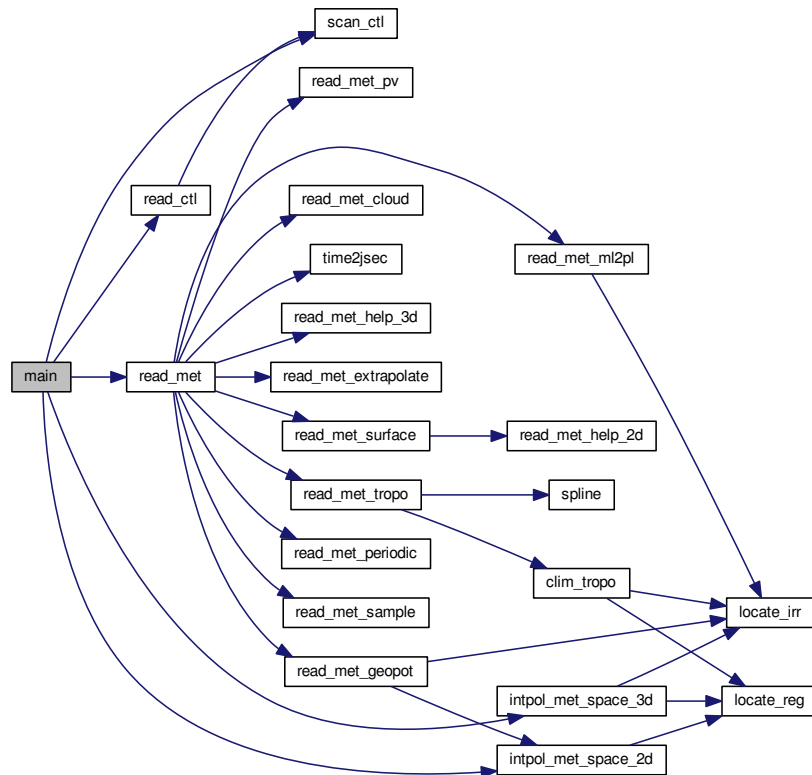
```

```

00158         if (gsl_finite(pt)) {
00159             ptm[iz] += pt;
00160             ztm[iz] += zt;
00161             ttm[iz] += tt;
00162             h2otm[iz] += h2ot;
00163             npt[iz]++;
00164         }
00165         np[iz]++;
00166     }
00167 }
00168 }
00169
00170 /* Create output file... */
00171 printf("Write meteorological data file: %s\n", argv[2]);
00172 if (!(out = fopen(argv[2], "w")))
00173     ERRMSG("Cannot create file!");
00174
00175 /* Write header... */
00176 fprintf(out,
00177     "# $1 = time [s]\n"
00178     "# $2 = altitude [km]\n"
00179     "# $3 = longitude [deg]\n"
00180     "# $4 = latitude [deg]\n"
00181     "# $5 = pressure [hPa]\n"
00182     "# $6 = temperature [K]\n"
00183     "# $7 = zonal wind [m/s]\n"
00184     "# $8 = meridional wind [m/s]\n"
00185     "# $9 = vertical wind [hPa/s]\n"
00186     "# $10 = H2O volume mixing ratio [ppv]\n");
00187 fprintf(out,
00188     "# $11 = O3 volume mixing ratio [ppv]\n"
00189     "# $12 = geopotential height [km]\n"
00190     "# $13 = potential vorticity [PVU]\n"
00191     "# $14 = surface pressure [hPa]\n"
00192     "# $15 = tropopause pressure [hPa]\n"
00193     "# $16 = tropopause geopotential height [km]\n"
00194     "# $17 = tropopause temperature [K]\n"
00195     "# $18 = tropopause water vapor [ppv]\n"
00196     "# $19 = cloud liquid water content [kg/kg]\n"
00197     "# $20 = cloud ice water content [kg/kg]\n");
00198 fprintf(out,
00199     "# $21 = total column cloud water [kg/m^2]\n"
00200     "# $22 = cloud top pressure [hPa]\n");
00201
00202 /* Write data... */
00203 for (iz = 0; iz < nz; iz++)
00204     fprintf(out,
00205         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00206         timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00207         latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00208         vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00209         o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00210         psm[iz] / np[iz], ptm[iz] / npt[iz], ztm[iz] / npt[iz],
00211         ttm[iz] / npt[iz], h2otm[iz] / npt[iz], lwcm[iz] / np[iz],
00212         iwcm[iz] / np[iz], clm[iz] / np[iz], pcm[iz] / np[iz]);
00213
00214 /* Close file... */
00215 fclose(out);
00216
00217 /* Free... */
00218 free(met);
00219
00220 return EXIT_SUCCESS;
00221 }

```

Here is the call graph for this function:



5.26 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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028  Dimensions...
00029  ----- */
00030
00032 #define NZ 1000
00033
00034 /* -----
00035  Main...
00036  ----- */
00037
00038 int main(
00039     int argc,
00040     char *argv[]) {

```

```

00041
00042     ctl_t ctl;
00043
00044     met_t *met;
00045
00046     FILE *out;
00047
00048     static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049         lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ], w,
00050         wm[NZ], h2o, h2om[NZ], h2ot, h2otm[NZ], o3, o3m[NZ], lwc, lwcm[NZ],
00051         iwc, iwcm[NZ], ps, psm[NZ], pt, ptm[NZ], pc, pcm[NZ], cl, clm[NZ],
00052         tt, ttm[NZ], zm[NZ], zt, ztm[NZ], pv, pvm[NZ], plev[NZ], cw[3];
00053
00054     static int i, iz, np[NZ], npt[NZ], nz, ci[3];
00055
00056     /* Allocate... */
00057     ALLOC(met, met_t, 1);
00058
00059     /* Check arguments... */
00060     if (argc < 4)
00061         ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00062
00063     /* Read control parameters... */
00064     read_ctl(argv[1], argc, argv, &ctl);
00065     z0 = scan_ctl(argv[1], argc, argv, "PROF_Z0", -1, "-999", NULL);
00066     z1 = scan_ctl(argv[1], argc, argv, "PROF_Z1", -1, "-999", NULL);
00067     dz = scan_ctl(argv[1], argc, argv, "PROF_DZ", -1, "-999", NULL);
00068     lon0 = scan_ctl(argv[1], argc, argv, "PROF_LON0", -1, "0", NULL);
00069     lon1 = scan_ctl(argv[1], argc, argv, "PROF_LON1", -1, "0", NULL);
00070     dlon = scan_ctl(argv[1], argc, argv, "PROF_DLON", -1, "-999", NULL);
00071     lat0 = scan_ctl(argv[1], argc, argv, "PROF_LAT0", -1, "0", NULL);
00072     lat1 = scan_ctl(argv[1], argc, argv, "PROF_LAT1", -1, "0", NULL);
00073     dlat = scan_ctl(argv[1], argc, argv, "PROF_DLAT", -1, "-999", NULL);
00074
00075     /* Loop over input files... */
00076     for (i = 3; i < argc; i++) {
00077
00078         /* Read meteorological data... */
00079         if (!read_met(&ctl, argv[i], met))
00080             continue;
00081
00082         /* Set vertical grid... */
00083         if (z0 < 0)
00084             z0 = Z(met->p[0]);
00085         if (z1 < 0)
00086             z1 = Z(met->p[met->np - 1]);
00087         nz = 0;
00088         if (dz < 0) {
00089             for (iz = 0; iz < met->np; iz++)
00090                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00091                     plev[nz] = met->p[iz];
00092                     if ((++nz) > NZ)
00093                         ERRMSG("Too many pressure levels!");
00094                 }
00095             } else
00096                 for (z = z0; z <= z1; z += dz) {
00097                     plev[nz] = P(z);
00098                     if ((++nz) > NZ)
00099                         ERRMSG("Too many pressure levels!");
00100                 }
00101
00102         /* Set horizontal grid... */
00103         if (dlon <= 0)
00104             dlon = fabs(met->lon[1] - met->lon[0]);
00105         if (dlat <= 0)
00106             dlat = fabs(met->lat[1] - met->lat[0]);
00107
00108         /* Average... */
00109         for (iz = 0; iz < nz; iz++)
00110             for (lon = lon0; lon <= lon1; lon += dlon)
00111                 for (lat = lat0; lat <= lat1; lat += dlat) {
00112
00113                     /* Interpolate meteo data... */
00114                     intpol_met_space_3d(met, met->z, plev[iz], lon, lat, &z, ci, cw, 1);
00115                     intpol_met_space_3d(met, met->t, plev[iz], lon, lat, &t, ci, cw, 0);
00116                     intpol_met_space_3d(met, met->u, plev[iz], lon, lat, &u, ci, cw, 0);
00117                     intpol_met_space_3d(met, met->v, plev[iz], lon, lat, &v, ci, cw, 0);
00118                     intpol_met_space_3d(met, met->w, plev[iz], lon, lat, &w, ci, cw, 0);
00119                     intpol_met_space_3d(met, met->pv, plev[iz], lon, lat, &pv, ci, cw,
00120                                         0);
00121                     intpol_met_space_3d(met, met->h2o, plev[iz], lon, lat, &h2o, ci, cw,
00122                                         0);
00123                     intpol_met_space_3d(met, met->o3, plev[iz], lon, lat, &o3, ci, cw,
00124                                         0);
00125                     intpol_met_space_3d(met, met->lwc, plev[iz], lon, lat, &lwc, ci, cw,
00126                                         0);
00127                     intpol_met_space_3d(met, met->iwc, plev[iz], lon, lat, &iwc, ci, cw,

```

```

00128         0);
00129     intpol_met_space_2d(met, met->ps, lon, lat, &ps, ci, cw, 0);
00130     intpol_met_space_2d(met, met->pt, lon, lat, &pt, ci, cw, 0);
00131     intpol_met_space_2d(met, met->pc, lon, lat, &pc, ci, cw, 0);
00132     intpol_met_space_2d(met, met->c1, lon, lat, &c1, ci, cw, 0);
00133
00134     /* Interpolate tropopause data... */
00135     intpol_met_space_3d(met, met->z, pt, lon, lat, &zt, ci, cw, 1);
00136     intpol_met_space_3d(met, met->t, pt, lon, lat, &tt, ci, cw, 0);
00137     intpol_met_space_3d(met, met->h2o, pt, lon, lat, &h2ot, ci, cw, 0);
00138
00139     /* Averaging... */
00140     if (gsl_finite(t) && gsl_finite(u)
00141         && gsl_finite(v) && gsl_finite(w)) {
00142         timem[iz] += met->time;
00143         lonm[iz] += lon;
00144         latm[iz] += lat;
00145         zm[iz] += z;
00146         tm[iz] += t;
00147         um[iz] += u;
00148         vm[iz] += v;
00149         wm[iz] += w;
00150         pvm[iz] += pv;
00151         h2om[iz] += h2o;
00152         o3m[iz] += o3;
00153         psm[iz] += ps;
00154         pcm[iz] += pc;
00155         clm[iz] += c1;
00156         lwcm[iz] += lwc;
00157         iwcm[iz] += iwc;
00158         if (gsl_finite(pt)) {
00159             ptm[iz] += pt;
00160             ztm[iz] += zt;
00161             ttm[iz] += tt;
00162             h2otm[iz] += h2ot;
00163             npt[iz]++;
00164         }
00165         np[iz]++;
00166     }
00167 }
00168 }
00169
00170 /* Create output file... */
00171 printf("Write meteorological data file: %s\n", argv[2]);
00172 if (!(out = fopen(argv[2], "w")))
00173     ERRMSG("Cannot create file!");
00174
00175 /* Write header... */
00176 fprintf(out,
00177     "# $1 = time [s]\n"
00178     "# $2 = altitude [km]\n"
00179     "# $3 = longitude [deg]\n"
00180     "# $4 = latitude [deg]\n"
00181     "# $5 = pressure [hPa]\n"
00182     "# $6 = temperature [K]\n"
00183     "# $7 = zonal wind [m/s]\n"
00184     "# $8 = meridional wind [m/s]\n"
00185     "# $9 = vertical wind [hPa/s]\n"
00186     "# $10 = H2O volume mixing ratio [ppv]\n");
00187 fprintf(out,
00188     "# $11 = O3 volume mixing ratio [ppv]\n"
00189     "# $12 = geopotential height [km]\n"
00190     "# $13 = potential vorticity [PVU]\n"
00191     "# $14 = surface pressure [hPa]\n"
00192     "# $15 = tropopause pressure [hPa]\n"
00193     "# $16 = tropopause geopotential height [km]\n"
00194     "# $17 = tropopause temperature [K]\n"
00195     "# $18 = tropopause water vapor [ppv]\n"
00196     "# $19 = cloud liquid water content [kg/kg]\n"
00197     "# $20 = cloud ice water content [kg/kg]\n");
00198 fprintf(out,
00199     "# $21 = total column cloud water [kg/m^2]\n"
00200     "# $22 = cloud top pressure [hPa]\n");
00201
00202 /* Write data... */
00203 for (iz = 0; iz < nz; iz++)
00204     fprintf(out,
00205         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00206         timem[iz] / np[iz], Z(plev[iz]), lonm[iz] / np[iz],
00207         latm[iz] / np[iz], plev[iz], tm[iz] / np[iz], um[iz] / np[iz],
00208         vm[iz] / np[iz], wm[iz] / np[iz], h2om[iz] / np[iz],
00209         o3m[iz] / np[iz], zm[iz] / np[iz], pvm[iz] / np[iz],
00210         psm[iz] / np[iz], ptm[iz] / npt[iz], ztm[iz] / npt[iz],
00211         ttm[iz] / npt[iz], h2otm[iz] / npt[iz], lwcm[iz] / np[iz],
00212         iwcm[iz] / np[iz], clm[iz] / np[iz], pcm[iz] / np[iz]);
00213
00214 /* Close file... */

```

```

00215     fclose(out);
00216
00217     /* Free... */
00218     free(met);
00219
00220     return EXIT_SUCCESS;
00221 }

```

5.27 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

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

5.27.1 Detailed Description

Sample meteorological data at given geolocations.

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

5.27.2 Function Documentation

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

Definition at line 31 of file [met_sample.c](#).

```

00033     {
00034
00035     ctl_t ctl;
00036
00037     atm_t *atm;
00038
00039     met_t *met0, *met1;
00040
00041     FILE *out;
00042
00043     double h2o, h2ot, o3, lwc, iwc, p0, p1, pref, ps, pt, pc, cl, pv, t, tt, u,
00044         v, w, z, zm, zref, zt, cw[3];
00045
00046     int geopot, ip, it, ci[3];
00047
00048     /* Check arguments... */
00049     if (argc < 4)
00050         ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");
00051
00052     /* Allocate... */
00053     ALLOC(atm, atm_t, 1);
00054     ALLOC(met0, met_t, 1);
00055     ALLOC(met1, met_t, 1);
00056
00057     /* Read control parameters... */
00058     read_ctl(argv[1], argc, argv, &ctl);
00059     geopot =
00060         (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00061
00062     /* Read atmospheric data... */
00063     if (!read_atm(argv[4], &ctl, atm))
00064         ERRMSG("Cannot open file!");
00065
00066     /* Create output file... */
00067     printf("Write meteorological data file: %s\n", argv[2]);
00068     if (!(out = fopen(argv[2], "w")))
00069         ERRMSG("Cannot create file!");

```

```

00070
00071 /* Write header... */
00072 fprintf(out,
00073     "# $1 = time [s]\n"
00074     "# $2 = altitude [km]\n"
00075     "# $3 = longitude [deg]\n"
00076     "# $4 = latitude [deg]\n"
00077     "# $5 = pressure [hPa]\n"
00078     "# $6 = temperature [K]\n"
00079     "# $7 = zonal wind [m/s]\n"
00080     "# $8 = meridional wind [m/s]\n"
00081     "# $9 = vertical wind [hPa/s]\n"
00082     "# $10 = H2O volume mixing ratio [ppv]\n");
00083 fprintf(out,
00084     "# $11 = O3 volume mixing ratio [ppv]\n"
00085     "# $12 = geopotential height [km]\n"
00086     "# $13 = potential vorticity [PVU]\n"
00087     "# $14 = surface pressure [hPa]\n"
00088     "# $15 = tropopause pressure [hPa]\n"
00089     "# $16 = tropopause geopotential height [km]\n"
00090     "# $17 = tropopause temperature [K]\n"
00091     "# $18 = tropopause water vapor [ppv]\n"
00092     "# $19 = cloud liquid water content [kg/kg]\n"
00093     "# $20 = cloud ice water content [kg/kg]\n");
00094 fprintf(out,
00095     "# $21 = total column cloud water [kg/m^2]\n"
00096     "# $22 = cloud top pressure [hPa]\n");
00097
00098 /* Loop over air parcels... */
00099 for (ip = 0; ip < atm->np; ip++) {
00100
00101     /* Get meteorological data... */
00102     get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);
00103
00104     /* Set reference pressure for interpolation... */
00105     pref = atm->p[ip];
00106     if (geopot) {
00107         zref = Z(pref);
00108         p0 = met0->p[0];
00109         p1 = met0->p[met0->np - 1];
00110         for (it = 0; it < 24; it++) {
00111             pref = 0.5 * (p0 + p1);
00112             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip], pref,
00113                 atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
00114             if (zref > zm || !gsl_finite(zm))
00115                 p0 = pref;
00116             else
00117                 p1 = pref;
00118         }
00119         pref = 0.5 * (p0 + p1);
00120     }
00121
00122     /* Interpolate meteo data... */
00123     intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip], pref,
00124         atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
00125     intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip], pref,
00126         atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
00127     intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip], pref,
00128         atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
00129     intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip], pref,
00130         atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
00131     intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip], pref,
00132         atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
00133     intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
time[ip], pref,
00134         atm->lon[ip], atm->lat[ip], &pv, ci, cw, 0);
00135     intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip], pref,
00136         atm->lon[ip], atm->lat[ip], &h2o, ci, cw, 0);
00137     intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->
time[ip], pref,
00138         atm->lon[ip], atm->lat[ip], &o3, ci, cw, 0);
00139     intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip], pref,
00140         atm->lon[ip], atm->lat[ip], &lwc, ci, cw, 0);
00141     intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip], pref,
00142         atm->lon[ip], atm->lat[ip], &iwc, ci, cw, 0);
00143     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
00144         atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);

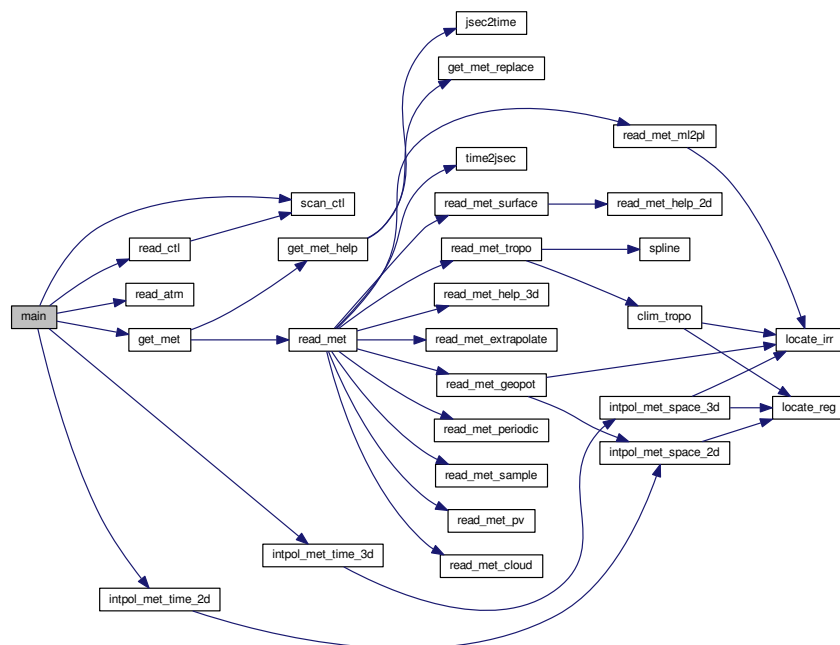
```

```

00145     intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
time[ip],
00146                     atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
00147     intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
00148                     atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
00149     intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
00150                     atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
00151
00152     /* Interpolate tropopause data... */
00153     intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip], pt,
00154                     atm->lon[ip], atm->lat[ip], &zt, ci, cw, 1);
00155     intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip], pt,
00156                     atm->lon[ip], atm->lat[ip], &tt, ci, cw, 0);
00157     intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip], pt,
00158                     atm->lon[ip], atm->lat[ip], &h2ot, ci, cw, 0);
00159
00160     /* Write data... */
00161     fprintf(out,
00162         "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00163         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00164         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt, h2ot, lwc,
00165         iwc, cl, pc);
00166 }
00167
00168 /* Close file... */
00169 fclose(out);
00170
00171 /* Free... */
00172 free(atm);
00173 free(met0);
00174 free(met1);
00175
00176 return EXIT_SUCCESS;
00177 }

```

Here is the call graph for this function:



5.28 met_sample.c

```
00001 /*
```



```

00002  This file is part of MPTRAC.
00003
00004  MPTRAC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  MPTRAC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2013-2019 Forschungszentrum Juelich GmbH
00018  */
00019
00025  #include "libtrac.h"
00026
00027  /* -----
00028  Main...
00029  ----- */
00030
00031  int main(
00032  int argc,
00033  char *argv[]) {
00034
00035  ctl_t ctl;
00036
00037  atm_t *atm;
00038
00039  met_t *met0, *met1;
00040
00041  FILE *out;
00042
00043  double h2o, h2ot, o3, lwc, iwc, p0, pl, pref, ps, pt, pc, cl, pv, t, tt, u,
00044  v, w, z, zm, zref, zt, cw[3];
00045
00046  int geopot, ip, it, ci[3];
00047
00048  /* Check arguments... */
00049  if (argc < 4)
00050      ERRMSG("Give parameters: <ctl> <sample.tab> <metbase> <atm_in>");
00051
00052  /* Allocate... */
00053  ALLOC(atm, atm_t, 1);
00054  ALLOC(met0, met_t, 1);
00055  ALLOC(met1, met_t, 1);
00056
00057  /* Read control parameters... */
00058  read_ctl(argv[1], argc, argv, &ctl);
00059  geopot =
00060      (int) scan_ctl(argv[1], argc, argv, "SAMPLE_GEOPOT", -1, "0", NULL);
00061
00062  /* Read atmospheric data... */
00063  if (!read_atm(argv[4], &ctl, atm))
00064      ERRMSG("Cannot open file!");
00065
00066  /* Create output file... */
00067  printf("Write meteorological data file: %s\n", argv[2]);
00068  if (!(out = fopen(argv[2], "w")))
00069      ERRMSG("Cannot create file!");
00070
00071  /* Write header... */
00072  fprintf(out,
00073  "# $1 = time [s]\n"
00074  "# $2 = altitude [km]\n"
00075  "# $3 = longitude [deg]\n"
00076  "# $4 = latitude [deg]\n"
00077  "# $5 = pressure [hPa]\n"
00078  "# $6 = temperature [K]\n"
00079  "# $7 = zonal wind [m/s]\n"
00080  "# $8 = meridional wind [m/s]\n"
00081  "# $9 = vertical wind [hPa/s]\n"
00082  "# $10 = H2O volume mixing ratio [ppv]\n");
00083  fprintf(out,
00084  "# $11 = O3 volume mixing ratio [ppv]\n"
00085  "# $12 = geopotential height [km]\n"
00086  "# $13 = potential vorticity [PVU]\n"
00087  "# $14 = surface pressure [hPa]\n"
00088  "# $15 = tropopause pressure [hPa]\n"
00089  "# $16 = tropopause geopotential height [km]\n"
00090  "# $17 = tropopause temperature [K]\n"
00091  "# $18 = tropopause water vapor [ppv]\n"
00092  "# $19 = cloud liquid water content [kg/kg]\n"
00093  "# $20 = cloud ice water content [kg/kg]\n");

```

```

00094     fprintf(out,
00095         "## $21 = total column cloud water [kg/m^2]\n"
00096         "## $22 = cloud top pressure [hPa]\n\n");
00097
00098     /* Loop over air parcels... */
00099     for (ip = 0; ip < atm->np; ip++) {
00100
00101         /* Get meteorological data... */
00102         get_met(&ctl, argv[3], atm->time[ip], &met0, &met1);
00103
00104         /* Set reference pressure for interpolation... */
00105         pref = atm->p[ip];
00106         if (geopot) {
00107             zref = Z(pref);
00108             p0 = met0->p[0];
00109             p1 = met0->p[met0->np - 1];
00110             for (it = 0; it < 24; it++) {
00111                 pref = 0.5 * (p0 + p1);
00112                 intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
00113 time[ip], pref,
00114                                     atm->lon[ip], atm->lat[ip], &zm, ci, cw, 1);
00115                 if (zref > zm || !gsl_finite(zm))
00116                     p0 = pref;
00117                 else
00118                     p1 = pref;
00119                 pref = 0.5 * (p0 + p1);
00120             }
00121
00122             /* Interpolate meteo data... */
00123             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
00124 time[ip], pref,
00125                                     atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
00126             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
00127 time[ip], pref,
00128                                     atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
00129             intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
00130 time[ip], pref,
00131                                     atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
00132             intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
00133 time[ip], pref,
00134                                     atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
00135             intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
00136 time[ip], pref,
00137                                     atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
00138             intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
00139 time[ip], pref,
00140                                     atm->lon[ip], atm->lat[ip], &pv, ci, cw, 0);
00141             intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
00142 time[ip], pref,
00143                                     atm->lon[ip], atm->lat[ip], &h2o, ci, cw, 0);
00144             intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->
00145 time[ip], pref,
00146                                     atm->lon[ip], atm->lat[ip], &o3, ci, cw, 0);
00147             intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
00148 time[ip], pref,
00149                                     atm->lon[ip], atm->lat[ip], &lwc, ci, cw, 0);
00150             intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
00151 time[ip], pref,
00152                                     atm->lon[ip], atm->lat[ip], &iwc, ci, cw, 0);
00153             intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
00154 time[ip],
00155                                     atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
00156             intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
00157 time[ip],
00158                                     atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
00159             intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
00160 time[ip],
00161                                     atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
00162             intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
00163 time[ip],
00164                                     atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
00165
00166             /* Interpolate tropopause data... */
00167             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
00168 time[ip], pt,
00169                                     atm->lon[ip], atm->lat[ip], &zt, ci, cw, 1);
00170             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
00171 time[ip], pt,
00172                                     atm->lon[ip], atm->lat[ip], &tt, ci, cw, 0);
00173             intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
00174 time[ip], pt,
00175                                     atm->lon[ip], atm->lat[ip], &h2ot, ci, cw, 0);
00176
00177             /* Write data... */
00178             fprintf(out,
00179                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",

```

```

00163         atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00164         atm->p[ip], t, u, v, w, h2o, o3, z, pv, ps, pt, zt, tt, h2ot, lwc,
00165         iwc, cl, pc);
00166     }
00167
00168     /* Close file... */
00169     fclose(out);
00170
00171     /* Free... */
00172     free(atm);
00173     free(met0);
00174     free(met1);
00175
00176     return EXIT_SUCCESS;
00177 }

```

5.29 met_zm.c File Reference

Extract zonal mean from meteorological data.

Functions

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

5.29.1 Detailed Description

Extract zonal mean from meteorological data.

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

5.29.2 Function Documentation

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

Definition at line 41 of file [met_zm.c](#).

```

00043     {
00044
00045         ctl_t ctl;
00046
00047         met_t *met;
00048
00049         FILE *out;
00050
00051         static double timem[NZ][NY], psm[NZ][NY], ptm[NZ][NY], pcm[NZ][NY],
00052         clm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY],
00053         wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY], pvm[NZ][NY], o3m[NZ][NY],
00054         lwcm[NZ][NY], iwcm[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
00055         ps, pt, pc, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc, lat, lat0, lat1,
00056         dlat, lats[NY], cw[3];
00057
00058         static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz, ci[3];
00059
00060         /* Allocate... */
00061         ALLOC(met, met_t, 1);
00062
00063         /* Check arguments... */
00064         if (argc < 4)
00065             ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00066
00067         /* Read control parameters... */
00068         read_ctl(argv[1], argc, argv, &ctl);
00069         z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00070         z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00071         dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);

```

```

00072 lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00073 lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00074 dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00075
00076 /* Loop over files... */
00077 for (i = 3; i < argc; i++) {
00078
00079     /* Read meteorological data... */
00080     if (!read_met(&ctl, argv[i], met))
00081         continue;
00082
00083     /* Set vertical grid... */
00084     if (z0 < 0)
00085         z0 = Z(met->p[0]);
00086     if (z1 < 0)
00087         z1 = Z(met->p[met->np - 1]);
00088     nz = 0;
00089     if (dz < 0) {
00090         for (iz = 0; iz < met->np; iz++)
00091             if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00092                 plev[nz] = met->p[iz];
00093                 if (++nz > NZ)
00094                     ERRMSG("Too many pressure levels!");
00095             }
00096     } else
00097         for (z = z0; z <= z1; z += dz) {
00098             plev[nz] = P(z);
00099             if (++nz > NZ)
00100                 ERRMSG("Too many pressure levels!");
00101         }
00102
00103     /* Set horizontal grid... */
00104     if (dlat <= 0)
00105         dlat = fabs(met->lat[1] - met->lat[0]);
00106     ny = 0;
00107     if (lat0 < -90 && lat1 > 90) {
00108         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00109         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00110     }
00111     for (lat = lat0; lat <= lat1; lat += dlat) {
00112         lats[ny] = lat;
00113         if (++ny > NY)
00114             ERRMSG("Too many latitudes!");
00115     }
00116
00117     /* Average... */
00118     for (ix = 0; ix < met->nx; ix++)
00119         for (iy = 0; iy < ny; iy++)
00120             for (iz = 0; iz < nz; iz++) {
00121
00122                 /* Interpolate meteo data... */
00123                 lon[ix],
00124                     intpol_met_space_3d(met, met->z, plev[iz], met->
00125                     met->lat[iy], &z, ci, cw, 1);
00126                 lon[ix],
00127                     intpol_met_space_3d(met, met->t, plev[iz], met->
00128                     met->lat[iy], &t, ci, cw, 0);
00129                 lon[ix],
00130                     intpol_met_space_3d(met, met->u, plev[iz], met->
00131                     met->lat[iy], &u, ci, cw, 0);
00132                 lon[ix],
00133                     intpol_met_space_3d(met, met->v, plev[iz], met->
00134                     met->lat[iy], &v, ci, cw, 0);
00135                 lon[ix],
00136                     intpol_met_space_3d(met, met->w, plev[iz], met->
00137                     met->lat[iy], &w, ci, cw, 0);
00138                 lon[ix],
00139                     intpol_met_space_3d(met, met->pv, plev[iz], met->
00140                     met->lat[iy], &pv, ci, cw, 0);
00141                 lon[ix],
00142                     intpol_met_space_3d(met, met->h2o, plev[iz], met->
00143                     met->lat[iy], &h2o, ci, cw, 0);
00144                 lon[ix],
00145                     intpol_met_space_3d(met, met->o3, plev[iz], met->
00146                     met->lat[iy], &o3, ci, cw, 0);
00147                 lon[ix],
00148                     intpol_met_space_3d(met, met->lw, plev[iz], met->
00149                     met->lat[iy], &lw, ci, cw, 0);
00150                 lon[ix],
00151                     intpol_met_space_3d(met, met->lwc, plev[iz], met->
00152                     met->lat[iy], &lwc, ci, cw, 0);
00153                 lon[ix],
00154                     intpol_met_space_3d(met, met->iwc, plev[iz], met->
00155                     met->lat[iy], &iwc, ci, cw, 0);
00156                 lon[ix],
00157                     intpol_met_space_2d(met, met->ps, met->lon[ix], met->
00158                     lat[iy], &ps,
00159                     ci, cw, 0);
00160                 lon[ix],
00161                     intpol_met_space_2d(met, met->pt, met->lon[ix], met->
00162                     lat[iy], &pt,
00163                     ci, cw, 0);

```

```

00147         intpol_met_space_2d(met, met->pc, met->lon[ix], met->
lat[iy], &pc,
00148                             ci, cw, 0);
00149         intpol_met_space_2d(met, met->c1, met->lon[ix], met->
lat[iy], &c1,
00150                             ci, cw, 0);
00151
00152         /* Interpolate tropopause data... */
00153         intpol_met_space_3d(met, met->z, pt, met->lon[ix], met->
lat[iy],
00154                             &zt, ci, cw, 1);
00155         intpol_met_space_3d(met, met->t, pt, met->lon[ix], met->
lat[iy],
00156                             &tt, ci, cw, 0);
00157         intpol_met_space_3d(met, met->h2o, pt, met->lon[ix], met->
lat[iy],
00158                             &h2ot, ci, cw, 0);
00159
00160         /* Averaging... */
00161         timem[iz][iy] += met->time;
00162         zm[iz][iy] += z;
00163         tm[iz][iy] += t;
00164         um[iz][iy] += u;
00165         vm[iz][iy] += v;
00166         wm[iz][iy] += w;
00167         pvm[iz][iy] += pv;
00168         h2om[iz][iy] += h2o;
00169         o3m[iz][iy] += o3;
00170         lwcm[iz][iy] += lwc;
00171         iwcm[iz][iy] += iwc;
00172         psm[iz][iy] += ps;
00173         pcm[iz][iy] += pc;
00174         clm[iz][iy] += cl;
00175         if (gsl_finite(pt)) {
00176             ptm[iz][iy] += pt;
00177             ztm[iz][iy] += zt;
00178             ttm[iz][iy] += tt;
00179             h2otm[iz][iy] += h2ot;
00180             npt[iz][iy]++;
00181         }
00182         np[iz][iy]++;
00183     }
00184 }
00185
00186 /* Create output file... */
00187 printf("Write meteorological data file: %s\n", argv[2]);
00188 if (!(out = fopen(argv[2], "w")))
00189     ERRMSG("Cannot create file!");
00190
00191 /* Write header... */
00192 fprintf(out,
00193         "# $1 = time [s]\n"
00194         "# $2 = altitude [km]\n"
00195         "# $3 = longitude [deg]\n"
00196         "# $4 = latitude [deg]\n"
00197         "# $5 = pressure [hPa]\n"
00198         "# $6 = temperature [K]\n"
00199         "# $7 = zonal wind [m/s]\n"
00200         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00201 fprintf(out,
00202         "# $10 = H2O volume mixing ratio [ppv]\n"
00203         "# $11 = O3 volume mixing ratio [ppv]\n"
00204         "# $12 = geopotential height [km]\n"
00205         "# $13 = potential vorticity [PVU]\n"
00206         "# $14 = surface pressure [hPa]\n"
00207         "# $15 = tropopause pressure [hPa]\n"
00208         "# $16 = tropopause geopotential height [km]\n"
00209         "# $17 = tropopause temperature [K]\n"
00210         "# $18 = tropopause water vapor [ppv]\n"
00211         "# $19 = cloud liquid water content [kg/kg]\n"
00212         "# $20 = cloud ice water content [kg/kg]\n");
00213 fprintf(out,
00214         "# $21 = total column cloud water [kg/m^2]\n"
00215         "# $22 = cloud top pressure [hPa]\n");
00216
00217 /* Write data... */
00218 for (iz = 0; iz < nz; iz++) {
00219     fprintf(out, "\n");
00220     for (iy = 0; iy < ny; iy++)
00221         fprintf(out,
00222                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00223                 timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
00224                 plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00225                 vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00226                 h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00227                 zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00228                 psm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],

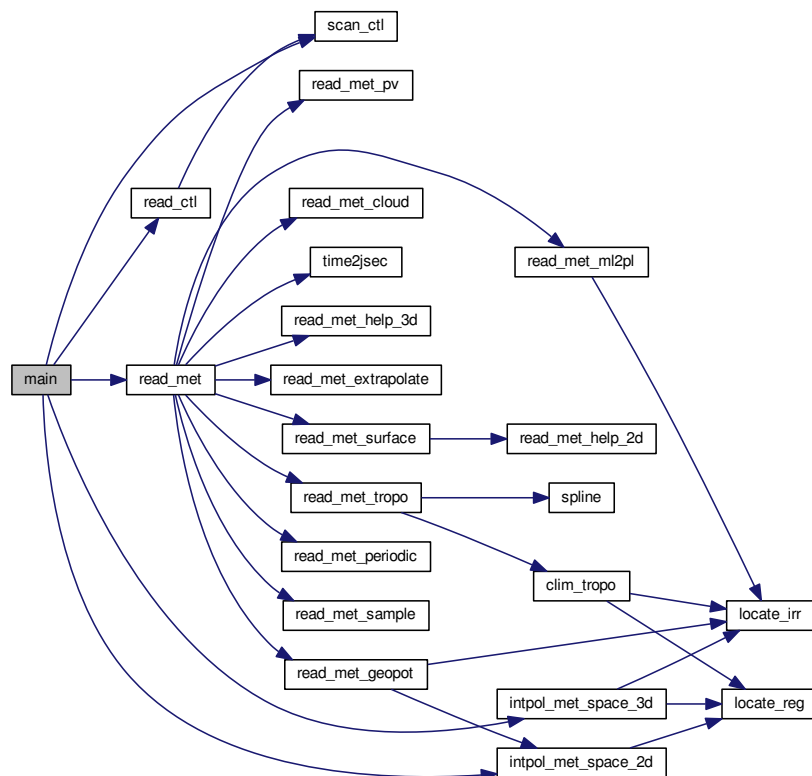
```

```

00229         ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
00230         h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00231         iwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
00232         pcm[iz][iy] / np[iz][iy]);
00233     }
00234
00235     /* Close file... */
00236     fclose(out);
00237
00238     /* Free... */
00239     free(met);
00240
00241     return EXIT_SUCCESS;
00242 }

```

Here is the call graph for this function:



5.30 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) 2013–2019 Forschungszentrum Juelich GmbH

```

```

00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028     Dimensions...
00029 ----- */
00030
00032 #define NZ 1000
00033
00035 #define NY 721
00036
00037 /* -----
00038     Main...
00039 ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     FILE *out;
00050
00051     static double timem[NZ][NY], psm[NZ][NY], ptm[NZ][NY], pcm[NZ][NY],
00052         clm[NZ][NY], ttm[NZ][NY], ztm[NZ][NY], tm[NZ][NY], um[NZ][NY], vm[NZ][NY],
00053         wm[NZ][NY], h2om[NZ][NY], h2otm[NZ][NY], pvm[NZ][NY], o3m[NZ][NY],
00054         lwcm[NZ][NY], iwc[NZ][NY], zm[NZ][NY], z, z0, z1, dz, zt, tt, plev[NZ],
00055         ps, pt, pc, cl, t, u, v, w, pv, h2o, h2ot, o3, lwc, iwc, lat, lat0, lat1,
00056         dlat, lats[NY], cw[3];
00057
00058     static int i, ix, iy, iz, np[NZ][NY], npt[NZ][NY], ny, nz, ci[3];
00059
00060     /* Allocate... */
00061     ALLOC(met, met_t, 1);
00062
00063     /* Check arguments... */
00064     if (argc < 4)
00065         ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00066
00067     /* Read control parameters... */
00068     read_ctl(argv[1], argc, argv, &ctl);
00069     z0 = scan_ctl(argv[1], argc, argv, "ZM_Z0", -1, "-999", NULL);
00070     z1 = scan_ctl(argv[1], argc, argv, "ZM_Z1", -1, "-999", NULL);
00071     dz = scan_ctl(argv[1], argc, argv, "ZM_DZ", -1, "-999", NULL);
00072     lat0 = scan_ctl(argv[1], argc, argv, "ZM_LAT0", -1, "-90", NULL);
00073     lat1 = scan_ctl(argv[1], argc, argv, "ZM_LAT1", -1, "90", NULL);
00074     dlat = scan_ctl(argv[1], argc, argv, "ZM_DLAT", -1, "-999", NULL);
00075
00076     /* Loop over files... */
00077     for (i = 3; i < argc; i++) {
00078
00079         /* Read meteorological data... */
00080         if (!read_met(&ctl, argv[i], met))
00081             continue;
00082
00083         /* Set vertical grid... */
00084         if (z0 < 0)
00085             z0 = Z(met->p[0]);
00086         if (z1 < 0)
00087             z1 = Z(met->p[met->np - 1]);
00088         nz = 0;
00089         if (dz < 0) {
00090             for (iz = 0; iz < met->np; iz++)
00091                 if (Z(met->p[iz]) >= z0 && Z(met->p[iz]) <= z1) {
00092                     plev[nz] = met->p[iz];
00093                     if ((++nz) > NZ)
00094                         ERRMSG("Too many pressure levels!");
00095                 }
00096             } else
00097                 for (z = z0; z <= z1; z += dz) {
00098                     plev[nz] = P(z);
00099                     if ((++nz) > NZ)
00100                         ERRMSG("Too many pressure levels!");
00101                 }
00102
00103         /* Set horizontal grid... */
00104         if (dlat <= 0)
00105             dlat = fabs(met->lat[1] - met->lat[0]);
00106         ny = 0;
00107         if (lat0 < -90 && lat1 > 90) {
00108             lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00109             lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00110         }
00111         for (lat = lat0; lat <= lat1; lat += dlat) {

```

```

00112     lats[ny] = lat;
00113     if (++ny > NY)
00114         ERRMSG("Too many latitudes!");
00115     }
00116
00117     /* Average... */
00118     for (ix = 0; ix < met->nx; ix++)
00119         for (iy = 0; iy < ny; iy++)
00120             for (iz = 0; iz < nz; iz++) {
00121
00122                 /* Interpolate meteo data... */
00123                 intpol_met_space_3d(met, met->z, plev[iz], met->
lon[ix],
00124                                     met->lat[iy], &z, ci, cw, 1);
00125                 intpol_met_space_3d(met, met->t, plev[iz], met->
lon[ix],
00126                                     met->lat[iy], &t, ci, cw, 0);
00127                 intpol_met_space_3d(met, met->u, plev[iz], met->
lon[ix],
00128                                     met->lat[iy], &u, ci, cw, 0);
00129                 intpol_met_space_3d(met, met->v, plev[iz], met->
lon[ix],
00130                                     met->lat[iy], &v, ci, cw, 0);
00131                 intpol_met_space_3d(met, met->w, plev[iz], met->
lon[ix],
00132                                     met->lat[iy], &w, ci, cw, 0);
00133                 intpol_met_space_3d(met, met->pv, plev[iz], met->
lon[ix],
00134                                     met->lat[iy], &pv, ci, cw, 0);
00135                 intpol_met_space_3d(met, met->h2o, plev[iz], met->
lon[ix],
00136                                     met->lat[iy], &h2o, ci, cw, 0);
00137                 intpol_met_space_3d(met, met->o3, plev[iz], met->
lon[ix],
00138                                     met->lat[iy], &o3, ci, cw, 0);
00139                 intpol_met_space_3d(met, met->lwc, plev[iz], met->
lon[ix],
00140                                     met->lat[iy], &lwc, ci, cw, 0);
00141                 intpol_met_space_3d(met, met->iwc, plev[iz], met->
lon[ix],
00142                                     met->lat[iy], &iwc, ci, cw, 0);
00143                 intpol_met_space_2d(met, met->ps, met->lon[ix], met->
lat[iy], &ps,
00144                                     ci, cw, 0);
00145                 intpol_met_space_2d(met, met->pt, met->lon[ix], met->
lat[iy], &pt,
00146                                     ci, cw, 0);
00147                 intpol_met_space_2d(met, met->pc, met->lon[ix], met->
lat[iy], &pc,
00148                                     ci, cw, 0);
00149                 intpol_met_space_2d(met, met->cl, met->lon[ix], met->
lat[iy], &cl,
00150                                     ci, cw, 0);
00151
00152                 /* Interpolate tropopause data... */
00153                 intpol_met_space_3d(met, met->z, pt, met->lon[ix], met->
lat[iy],
00154                                     &z, ci, cw, 1);
00155                 intpol_met_space_3d(met, met->t, pt, met->lon[ix], met->
lat[iy],
00156                                     &t, ci, cw, 0);
00157                 intpol_met_space_3d(met, met->h2o, pt, met->lon[ix], met->
lat[iy],
00158                                     &h2ot, ci, cw, 0);
00159
00160                 /* Averaging... */
00161                 timem[iz][iy] += met->time;
00162                 zm[iz][iy] += z;
00163                 tm[iz][iy] += t;
00164                 um[iz][iy] += u;
00165                 vm[iz][iy] += v;
00166                 wm[iz][iy] += w;
00167                 pvm[iz][iy] += pv;
00168                 h2om[iz][iy] += h2o;
00169                 o3m[iz][iy] += o3;
00170                 lwcm[iz][iy] += lwc;
00171                 iwcm[iz][iy] += iwc;
00172                 psm[iz][iy] += ps;
00173                 pcm[iz][iy] += pc;
00174                 clm[iz][iy] += cl;
00175                 if (gsl_finite(pt)) {
00176                     ptm[iz][iy] += pt;
00177                     ztm[iz][iy] += zt;
00178                     ttm[iz][iy] += tt;
00179                     h2otm[iz][iy] += h2ot;
00180                     npt[iz][iy]++;
00181                 }

```



```

00182         np[iz][iy]++;
00183     }
00184 }
00185
00186 /* Create output file... */
00187 printf("Write meteorological data file: %s\n", argv[2]);
00188 if (! (out = fopen(argv[2], "w")))
00189     ERRMSG("Cannot create file!");
00190
00191 /* Write header... */
00192 fprintf(out,
00193         "# $1 = time [s]\n"
00194         "# $2 = altitude [km]\n"
00195         "# $3 = longitude [deg]\n"
00196         "# $4 = latitude [deg]\n"
00197         "# $5 = pressure [hPa]\n"
00198         "# $6 = temperature [K]\n"
00199         "# $7 = zonal wind [m/s]\n"
00200         "# $8 = meridional wind [m/s]\n" "# $9 = vertical wind [hPa/s]\n");
00201 fprintf(out,
00202         "# $10 = H2O volume mixing ratio [ppv]\n"
00203         "# $11 = O3 volume mixing ratio [ppv]\n"
00204         "# $12 = geopotential height [km]\n"
00205         "# $13 = potential vorticity [PVU]\n"
00206         "# $14 = surface pressure [hPa]\n"
00207         "# $15 = tropopause pressure [hPa]\n"
00208         "# $16 = tropopause geopotential height [km]\n"
00209         "# $17 = tropopause temperature [K]\n"
00210         "# $18 = tropopause water vapor [ppv]\n"
00211         "# $19 = cloud liquid water content [kg/kg]\n"
00212         "# $20 = cloud ice water content [kg/kg]\n");
00213 fprintf(out,
00214         "# $21 = total column cloud water [kg/m^2]\n"
00215         "# $22 = cloud top pressure [hPa]\n");
00216
00217 /* Write data... */
00218 for (iz = 0; iz < nz; iz++) {
00219     fprintf(out, "\n");
00220     for (iy = 0; iy < ny; iy++)
00221         fprintf(out,
00222                 "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00223                 timem[iz][iy] / np[iz][iy], Z(plev[iz]), 0.0, lats[iy],
00224                 plev[iz], tm[iz][iy] / np[iz][iy], um[iz][iy] / np[iz][iy],
00225                 vm[iz][iy] / np[iz][iy], wm[iz][iy] / np[iz][iy],
00226                 h2om[iz][iy] / np[iz][iy], o3m[iz][iy] / np[iz][iy],
00227                 zm[iz][iy] / np[iz][iy], pvm[iz][iy] / np[iz][iy],
00228                 psm[iz][iy] / np[iz][iy], ptm[iz][iy] / npt[iz][iy],
00229                 ztm[iz][iy] / npt[iz][iy], ttm[iz][iy] / npt[iz][iy],
00230                 h2otm[iz][iy] / npt[iz][iy], lwcm[iz][iy] / np[iz][iy],
00231                 iwcm[iz][iy] / np[iz][iy], clm[iz][iy] / np[iz][iy],
00232                 pcm[iz][iy] / np[iz][iy]);
00233 }
00234
00235 /* Close file... */
00236 fclose(out);
00237
00238 /* Free... */
00239 free(met);
00240
00241 return EXIT_SUCCESS;
00242 }

```

5.31 nvtxmc.h File Reference

5.32 nvtxmc.h

```

00001 #define RED 0xFFFF0000
00002 #define BLUE 0xFF0000FF
00003 #define GREEN 0xFF008000
00004 #define YELLOW 0FFFFFFF0
00005 #define CYAN 0xFF00FFFF
00006 #define MAGENTA 0xFFFF00FF
00007 #define GRAY 0xFF808080
00008 #define PURPLE 0xFF800080
00009
00010 // Macro for calling nvtxRangePushEx
00011 #define RANGE_PUSH(range_title, range_color) { \
00012     nvtxEventAttributes_t eventAttrib = {0}; \
00013     eventAttrib.version = NVTX_VERSION; \
00014     eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE; \
00015     eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII; \

```

```

00016     eventAttrib.colorType = NVTX_COLOR_ARGB; \
00017     eventAttrib.color = range_color; \
00018     eventAttrib.message.ascii = range_title; \
00019     nvtxRangePushEx(&eventAttrib); \
00020 }
00021
00022 // Macro for calling nvtxRangePop
00023 #define RANGE_POP {\
00024     nvtxRangePop();\
00025 }

```

5.33 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

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

5.33.1 Detailed Description

Convert date to Julian seconds.

Definition in file [time2jsec.c](#).

5.33.2 Function Documentation

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

Definition at line 27 of file [time2jsec.c](#).

```

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

```

Here is the call graph for this function:



5.34 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 }

```

5.35 trac.c File Reference

Lagrangian particle dispersion model.

Functions

- void `module_advection` (`met_t` *met0, `met_t` *met1, `atm_t` *atm, double *dt)
Calculate advection of air parcels.
- void `module_decay` (`ctl_t` *ctl, `atm_t` *atm, double *dt)
Calculate exponential decay of particle mass.
- void `module_diffusion_init` (void)
Initialize random number generator...
- void `module_diffusion_meso` (`ctl_t` *ctl, `met_t` *met0, `met_t` *met1, `atm_t` *atm, `cache_t` *cache, double *dt, double *rs)
Calculate mesoscale diffusion.
- void `module_diffusion_rng` (double *rs, size_t n)
Generate random numbers.
- void `module_diffusion_turb` (`ctl_t` *ctl, `atm_t` *atm, double *dt, double *rs)

- Calculate turbulent diffusion.*
- void [module_isosurf_init](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache)
- Initialize isosurface module.*
- void [module_isosurf](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, [cache_t](#) *cache)
- Force air parcels to stay on isosurface.*
- void [module_meteo](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm)
- Interpolate meteorological data for air parcel positions.*
- void [module_position](#) ([met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- Check position of air parcels.*
- void [module_sedi](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- Calculate sedimentation of air parcels.*
- void [module_oh_chem](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- Calculate OH chemistry.*
- void [module_wet_deposition](#) ([ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double *dt)
- Calculate wet deposition.*
- void [write_output](#) (const char *dirname, [ctl_t](#) *ctl, [met_t](#) *met0, [met_t](#) *met1, [atm_t](#) *atm, double t)
- Write simulation output.*
- int [main](#) (int argc, char *argv[])

Variables

- [curandGenerator_t](#) [rng](#)

5.35.1 Detailed Description

Lagrangian particle dispersion model.

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

5.35.2 Function Documentation

5.35.2.1 void module_advection ([met_t](#) * *met0*, [met_t](#) * *met1*, [atm_t](#) * *atm*, double * *dt*)

Calculate advection of air parcels.

Definition at line 551 of file [trac.c](#).

```

00555         {
00556
00557     #ifdef _OPENACC
00558     #pragma acc data present (met0,met1,atm,dt)
00559     #pragma acc parallel loop independent gang vector
00560     #else
00561     #pragma omp parallel for default(shared)
00562     #endif
00563         for (int ip = 0; ip < atm->np; ip++)
00564             if (dt[ip] != 0) {
00565
00566                 int ci[3] = { 0 };
00567
00568                 double dtm = 0.0, v[3] = { 0.0 }, xm[3] = {
00569                     0.0 };
00570                 double cw[3] = { 0.0 };
00571
00572                 /* Interpolate meteorological data... */
00573                 intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],

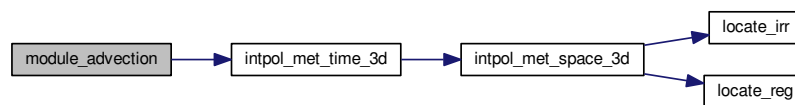
```

```

00574             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[0], ci,
00575             cw, 1);
00576     intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
00577             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[1], ci,
00578             cw, 0);
00579     intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
00580             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[2], ci,
00581             cw, 0);
00582
00583     /* Get position of the mid point... */
00584     dtm = atm->time[ip] + 0.5 * dt[ip];
00585     xm[0] =
00586         atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
00587     xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
00588     xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];
00589
00590     /* Interpolate meteorological data for mid point... */
00591     intpol_met_time_3d(met0, met0->u, met1, met1->u, dtm, xm[2], xm[0],
xm[1], &v[0], ci, cw, 1);
00592     intpol_met_time_3d(met0, met0->v, met1, met1->v, dtm, xm[2], xm[0],
xm[1], &v[1], ci, cw, 0);
00593     intpol_met_time_3d(met0, met0->w, met1, met1->w, dtm, xm[2], xm[0],
xm[1], &v[2], ci, cw, 0);
00594
00595     /* Save new position... */
00596     atm->time[ip] += dt[ip];
00597     atm->lon[ip] += DX2DEG(dt[ip] * v[0] / 1000., xm[1]);
00598     atm->lat[ip] += DY2DEG(dt[ip] * v[1] / 1000.);
00599     atm->p[ip] += dt[ip] * v[2];
00600 }
00601 }
00602 }
00603 }
00604 }

```

Here is the call graph for this function:



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

Calculate exponential decay of particle mass.

Definition at line 608 of file trac.c.

```

00611     {
00612
00613     /* Check quantity flags... */
00614     if (ctl->qnt_m < 0)
00615         ERRMSG("Module needs quantity mass!");
00616
00617     #ifdef _OPENACC
00618     #pragma acc data present(ctl,atm,dt)
00619     #pragma acc parallel loop independent gang vector
00620     #else
00621     #pragma omp parallel for default(shared)
00622     #endif
00623     for (int ip = 0; ip < atm->np; ip++)
00624         if (dt[ip] != 0) {
00625
00626             double p0, p1, pt, tdec, w;
00627
00628             /* Get tropopause pressure... */
00629             pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00630
00631             /* Get weighting factor... */
00632             p1 = pt * 0.866877899;

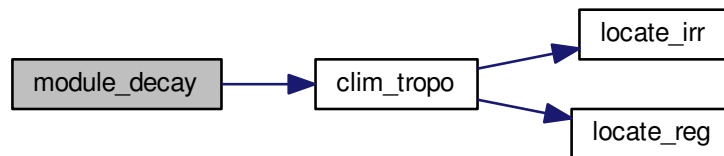
```

```

00633     p0 = pt / 0.866877899;
00634     if (atm->p[ip] > p0)
00635         w = 1;
00636     else if (atm->p[ip] < p1)
00637         w = 0;
00638     else
00639         w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00640
00641     /* Set lifetime... */
00642     tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00643
00644     /* Calculate exponential decay... */
00645     atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00646 }
00647 }

```

Here is the call graph for this function:



5.35.2.3 void module_diffusion_init (void)

Initialize random number generator...

Definition at line 651 of file [trac.c](#).

```

00652     {
00653
00654     /* Initialize random number generator... */
00655     #ifdef _OPENACC
00656
00657     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
00658         != CURAND_STATUS_SUCCESS)
00659         ERRMSG("Cannot create random number generator!");
00660     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
00661         != CURAND_STATUS_SUCCESS)
00662         ERRMSG("Cannot set stream for random number generator!");
00663
00664     #else
00665
00666     gsl_rng_env_setup();
00667     if (omp_get_max_threads() > NTHREADS)
00668         ERRMSG("Too many threads!");
00669     for (int i = 0; i < NTHREADS; i++) {
00670         rng[i] = gsl_rng_alloc(gsl_rng_default);
00671         gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00672     }
00673
00674     #endif
00675 }

```

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

Calculate mesoscale diffusion.

Definition at line 679 of file [trac.c](#).

```

00686         {
00687
00688 #ifdef _OPENACC
00689 #pragma acc data present(ctl,met0,met1,atm,cache,dt,rs)
00690 #pragma acc parallel loop independent gang vector
00691 #else
00692 #pragma omp parallel for default(shared)
00693 #endif
00694     for (int ip = 0; ip < atm->np; ip++)
00695         if (dt[ip] != 0) {
00696
00697             double u[16], v[16], w[16];
00698
00699             /* Get indices... */
00700             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00701             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00702             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00703
00704             /* Caching of wind standard deviations... */
00705             if (cache->tsig[ix][iy][iz] != met0->time) {
00706
00707                 /* Collect local wind data... */
00708                 u[0] = met0->u[ix][iy][iz];
00709                 u[1] = met0->u[ix + 1][iy][iz];
00710                 u[2] = met0->u[ix][iy + 1][iz];
00711                 u[3] = met0->u[ix + 1][iy + 1][iz];
00712                 u[4] = met0->u[ix][iy][iz + 1];
00713                 u[5] = met0->u[ix + 1][iy][iz + 1];
00714                 u[6] = met0->u[ix][iy + 1][iz + 1];
00715                 u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00716
00717                 v[0] = met0->v[ix][iy][iz];
00718                 v[1] = met0->v[ix + 1][iy][iz];
00719                 v[2] = met0->v[ix][iy + 1][iz];
00720                 v[3] = met0->v[ix + 1][iy + 1][iz];
00721                 v[4] = met0->v[ix][iy][iz + 1];
00722                 v[5] = met0->v[ix + 1][iy][iz + 1];
00723                 v[6] = met0->v[ix][iy + 1][iz + 1];
00724                 v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00725
00726                 w[0] = met0->w[ix][iy][iz];
00727                 w[1] = met0->w[ix + 1][iy][iz];
00728                 w[2] = met0->w[ix][iy + 1][iz];
00729                 w[3] = met0->w[ix + 1][iy + 1][iz];
00730                 w[4] = met0->w[ix][iy][iz + 1];
00731                 w[5] = met0->w[ix + 1][iy][iz + 1];
00732                 w[6] = met0->w[ix][iy + 1][iz + 1];
00733                 w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00734
00735                 /* Collect local wind data... */
00736                 u[8] = met1->u[ix][iy][iz];
00737                 u[9] = met1->u[ix + 1][iy][iz];
00738                 u[10] = met1->u[ix][iy + 1][iz];
00739                 u[11] = met1->u[ix + 1][iy + 1][iz];
00740                 u[12] = met1->u[ix][iy][iz + 1];
00741                 u[13] = met1->u[ix + 1][iy][iz + 1];
00742                 u[14] = met1->u[ix][iy + 1][iz + 1];
00743                 u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00744
00745                 v[8] = met1->v[ix][iy][iz];
00746                 v[9] = met1->v[ix + 1][iy][iz];
00747                 v[10] = met1->v[ix][iy + 1][iz];
00748                 v[11] = met1->v[ix + 1][iy + 1][iz];
00749                 v[12] = met1->v[ix][iy][iz + 1];
00750                 v[13] = met1->v[ix + 1][iy][iz + 1];
00751                 v[14] = met1->v[ix][iy + 1][iz + 1];
00752                 v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00753
00754                 w[8] = met1->w[ix][iy][iz];
00755                 w[9] = met1->w[ix + 1][iy][iz];
00756                 w[10] = met1->w[ix][iy + 1][iz];
00757                 w[11] = met1->w[ix + 1][iy + 1][iz];
00758                 w[12] = met1->w[ix][iy][iz + 1];
00759                 w[13] = met1->w[ix + 1][iy][iz + 1];
00760                 w[14] = met1->w[ix][iy + 1][iz + 1];
00761                 w[15] = met1->w[ix + 1][iy + 1][iz + 1];

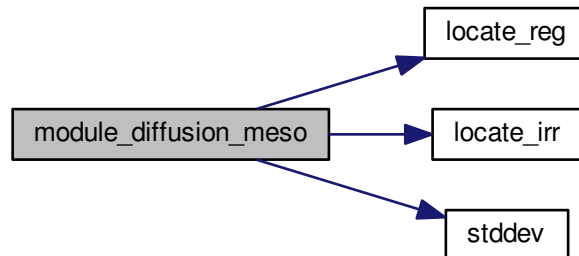
```

```

00762
00763     /* Get standard deviations of local wind data... */
00764     cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00765     cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00766     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00767     cache->tsig[ix][iy][iz] = met0->time;
00768 }
00769
00770 /* Set temporal correlations for mesoscale fluctuations... */
00771 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00772 double r2 = sqrt(1 - r * r);
00773
00774 /* Calculate horizontal mesoscale wind fluctuations... */
00775 if (ctl->turb_mesox > 0) {
00776     cache->up[ip] = (float)
00777         (r * cache->up[ip]
00778          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00779     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00780
00781     cache->vp[ip] = (float)
00782         (r * cache->vp[ip]
00783          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00784     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00785 }
00786
00787 /* Calculate vertical mesoscale wind fluctuations... */
00788 if (ctl->turb_mesoz > 0) {
00789     cache->wp[ip] = (float)
00790         (r * cache->wp[ip]
00791          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00792     atm->p[ip] += cache->wp[ip] * dt[ip];
00793 }
00794 }
00795 }

```

Here is the call graph for this function:



5.35.2.5 void module_diffusion_rng (double * rs, size_t n)

Generate random numbers.

Definition at line 799 of file [trac.c](#).

```

00801     {
00802
00803     #ifdef _OPENACC
00804
00805     #pragma acc host_data use_device(rs)
00806     {
00807         if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
00808             != CURAND_STATUS_SUCCESS)
00809             ERRMSG("Cannot create random numbers!");
00810     }

```



```

00811
00812 #else
00813
00814 #pragma omp parallel for default(shared)
00815     for (size_t i = 0; i < n; ++i)
00816         rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
00817
00818 #endif
00819
00820 }

```

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

Calculate turbulent diffusion.

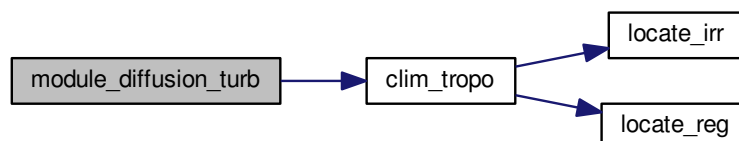
Definition at line 824 of file [trac.c](#).

```

00828     {
00829
00830 #ifdef _OPENACC
00831 #pragma acc data present(ctl,atm,dt,rs)
00832 #pragma acc parallel loop independent gang vector
00833 #else
00834 #pragma omp parallel for default(shared)
00835 #endif
00836     for (int ip = 0; ip < atm->np; ip++)
00837         if (dt[ip] != 0) {
00838
00839             double w;
00840
00841             /* Get tropopause pressure... */
00842             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00843
00844             /* Get weighting factor... */
00845             double p1 = pt * 0.866877899;
00846             double p0 = pt / 0.866877899;
00847             if (atm->p[ip] > p0)
00848                 w = 1;
00849             else if (atm->p[ip] < p1)
00850                 w = 0;
00851             else
00852                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00853
00854             /* Set diffusivity... */
00855             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
turb_dx_strat;
00856             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
turb_dz_strat;
00857
00858             /* Horizontal turbulent diffusion... */
00859             if (dx > 0) {
00860                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00861                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00862                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00863             }
00864
00865             /* Vertical turbulent diffusion... */
00866             if (dz > 0) {
00867                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00868                 atm->p[ip]
00869                     += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00870             }
00871         }
00872     }

```

Here is the call graph for this function:



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

Initialize isosurface module.

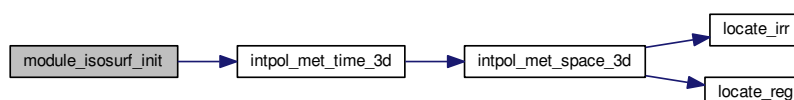
Definition at line 876 of file [trac.c](#).

```

00881         {
00882
00883     FILE *in;
00884
00885     char line[LEN];
00886
00887     double t, cw[3];
00888
00889     int ci[3];
00890
00891     /* Save pressure... */
00892     if (ctl->isosurf == 1)
00893         for (int ip = 0; ip < atm->np; ip++)
00894             cache->iso_var[ip] = atm->p[ip];
00895
00896     /* Save density... */
00897     else if (ctl->isosurf == 2)
00898         for (int ip = 0; ip < atm->np; ip++) {
00899             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00900                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00901                             1);
00902             cache->iso_var[ip] = atm->p[ip] / t;
00903         }
00904
00905     /* Save potential temperature... */
00906     else if (ctl->isosurf == 3)
00907         for (int ip = 0; ip < atm->np; ip++) {
00908             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00909                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00910                             1);
00911             cache->iso_var[ip] = THETA(atm->p[ip], t);
00912         }
00913
00914     /* Read balloon pressure data... */
00915     else if (ctl->isosurf == 4) {
00916
00917         /* Write info... */
00918         printf("Read balloon pressure data: %s\n", ctl->balloon);
00919
00920         /* Open file... */
00921         if (!(in = fopen(ctl->balloon, "r")))
00922             ERRMSG("Cannot open file!");
00923
00924         /* Read pressure time series... */
00925         while (fgets(line, LEN, in))
00926             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00927                     &(cache->iso_ps[cache->iso_n])) == 2)
00928                 if (++cache->iso_n > NP)
00929                     ERRMSG("Too many data points!");
00930
00931         /* Check number of points... */
00932         if (cache->iso_n < 1)
00933             ERRMSG("Could not read any data!");
00934
00935         /* Close file... */
00936         fclose(in);
00937     }
00938 }

```

Here is the call graph for this function:



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

Force air parcels to stay on isosurface.

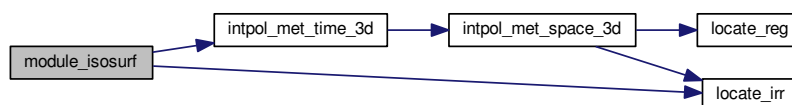
Definition at line 942 of file [trac.c](#).

```

00947     {
00948
00949     #ifdef _OPENACC
00950     #pragma acc data present(ctl,met0,met1,atm,cache)
00951     #pragma acc parallel loop independent gang vector
00952     #else
00953     #pragma omp parallel for default(shared)
00954     #endif
00955     for (int ip = 0; ip < atm->np; ip++) {
00956
00957         double t, cw[3];
00958
00959         int ci[3];
00960
00961         /* Restore pressure... */
00962         if (ctl->isosurf == 1)
00963             atm->p[ip] = cache->iso_var[ip];
00964
00965         /* Restore density... */
00966         else if (ctl->isosurf == 2) {
00967             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00968                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00969                             1);
00970             atm->p[ip] = cache->iso_var[ip] * t;
00971         }
00972
00973         /* Restore potential temperature... */
00974         else if (ctl->isosurf == 3) {
00975             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00976                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00977                             1);
00978             atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00979         }
00980
00981         /* Interpolate pressure... */
00982         else if (ctl->isosurf == 4) {
00983             if (atm->time[ip] <= cache->iso_ts[0])
00984                 atm->p[ip] = cache->iso_ps[0];
00985             else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00986                 atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00987             else {
00988                 int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
00989                 atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
00990                                cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00991                                atm->time[ip]);
00992             }
00993         }
00994     }
00995 }

```

Here is the call graph for this function:



5.35.2.9 void module_meteo (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*)

Interpolate meteorological data for air parcel positions.

Definition at line 999 of file [trac.c](#).

```

01003         {
01004
01005         /* Check quantity flags... */
01006         if (ctl->qnt_tsts >= 0)
01007             if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01008                 ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01009
01010 #ifdef _OPENACC
01011 #pragma acc data present(ctl,met0,met1,atm)
01012 #pragma acc parallel loop independent gang vector
01013 #else
01014 #pragma omp parallel for default(shared)
01015 #endif
01016         for (int ip = 0; ip < atm->np; ip++) {
01017
01018             double ps, pt, pc, pv, t, u, v, w, h2o, o3, lwc, iwc, z, cw[3];
01019
01020             int ci[3];
01021
01022             /* Interpolate meteorological data... */
01023             intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip],
01024                             atm->p[ip], atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
01025             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01026                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
01027             intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
01028                             atm->p[ip], atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
01029             intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
01030                             atm->p[ip], atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
01031             intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
01032                             atm->p[ip], atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
01033             intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
time[ip],
01034                             atm->p[ip], atm->lon[ip], atm->lat[ip], &pv, ci, cw,
0);
01035
01036             intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip],
01037                             atm->p[ip], atm->lon[ip], atm->lat[ip], &h2o, ci, cw,
0);
01038
01039             intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->
time[ip],
01040                             atm->p[ip], atm->lon[ip], atm->lat[ip], &o3, ci, cw,
0);
01041
01042             intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
01043                             atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
0);
01044
01045             intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
01046                             atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
0);
01047
01048             intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
01049                             atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
01050             intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
time[ip],
01051                             atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
01052             intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01053                             atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
01054
01055             /* Set surface pressure... */
01056             if (ctl->qnt_ps >= 0)
01057                 atm->q[ctl->qnt_ps][ip] = ps;
01058
01059             /* Set tropopause pressure... */
01060             if (ctl->qnt_pt >= 0)
01061                 atm->q[ctl->qnt_pt][ip] = pt;
01062
01063             /* Set pressure... */
01064             if (ctl->qnt_p >= 0)
01065                 atm->q[ctl->qnt_p][ip] = atm->p[ip];
01066

```

```

01067      /* Set geopotential height... */
01068      if (ctl->qnt_z >= 0)
01069          atm->q[ctl->qnt_z][ip] = z;
01070
01071      /* Set temperature... */
01072      if (ctl->qnt_t >= 0)
01073          atm->q[ctl->qnt_t][ip] = t;
01074
01075      /* Set zonal wind... */
01076      if (ctl->qnt_u >= 0)
01077          atm->q[ctl->qnt_u][ip] = u;
01078
01079      /* Set meridional wind... */
01080      if (ctl->qnt_v >= 0)
01081          atm->q[ctl->qnt_v][ip] = v;
01082
01083      /* Set vertical velocity... */
01084      if (ctl->qnt_w >= 0)
01085          atm->q[ctl->qnt_w][ip] = w;
01086
01087      /* Set water vapor vmr... */
01088      if (ctl->qnt_h2o >= 0)
01089          atm->q[ctl->qnt_h2o][ip] = h2o;
01090
01091      /* Set ozone vmr... */
01092      if (ctl->qnt_o3 >= 0)
01093          atm->q[ctl->qnt_o3][ip] = o3;
01094
01095      /* Set cloud liquid water content... */
01096      if (ctl->qnt_lwc >= 0)
01097          atm->q[ctl->qnt_lwc][ip] = lwc;
01098
01099      /* Set cloud ice water content... */
01100      if (ctl->qnt_iwc >= 0)
01101          atm->q[ctl->qnt_iwc][ip] = iwc;
01102
01103      /* Set cloud top pressure... */
01104      if (ctl->qnt_pc >= 0)
01105          atm->q[ctl->qnt_pc][ip] = pc;
01106
01107      /* Set nitric acid vmr... */
01108      if (ctl->qnt_hno3 >= 0)
01109          atm->q[ctl->qnt_hno3][ip] =
01110              clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]);
01111
01112      /* Set hydroxyl number concentration... */
01113      if (ctl->qnt_oh >= 0)
01114          atm->q[ctl->qnt_oh][ip] =
01115              clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]);
01116
01117      /* Calculate horizontal wind... */
01118      if (ctl->qnt_vh >= 0)
01119          atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
01120
01121      /* Calculate vertical velocity... */
01122      if (ctl->qnt_vz >= 0)
01123          atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
01124
01125      /* Calculate relative humidity... */
01126      if (ctl->qnt_rh >= 0)
01127          atm->q[ctl->qnt_rh][ip] = RH(atm->p[ip], t, h2o);
01128
01129      /* Calculate potential temperature... */
01130      if (ctl->qnt_theta >= 0)
01131          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
01132
01133      /* Set potential vorticity... */
01134      if (ctl->qnt_pv >= 0)
01135          atm->q[ctl->qnt_pv][ip] = pv;
01136
01137      /* Calculate T_ice (Marti and Mauersberger, 1993)... */
01138      if (ctl->qnt_tice >= 0)
01139          atm->q[ctl->qnt_tice][ip] =
01140              -2663.5 /
01141              (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
01142               12.537);
01143
01144      /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
01145      if (ctl->qnt_tnat >= 0) {
01146          double p_hno3;
01147          if (ctl->psc_hno3 > 0)
01148              p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
01149          else
01150              p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
01151                  * 1e-9 * atm->p[ip] / 1.333224;
01152          double p_h2o =
01153              (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;

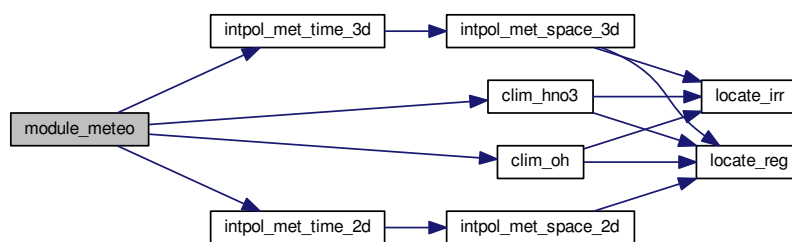
```

```

01154     double a = 0.009179 - 0.00088 * log10(p_h2o);
01155     double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01156     double c = -11397.0 / a;
01157     double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
01158     double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01159     if (x1 > 0)
01160         atm->q[ctl->qnt_tnat][ip] = x1;
01161     if (x2 > 0)
01162         atm->q[ctl->qnt_tnat][ip] = x2;
01163 }
01164
01165 /* Calculate T_STS (mean of T_ice and T_NAT)... */
01166 if (ctl->qnt_tsts >= 0)
01167     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
01168                                     + atm->q[ctl->qnt_tnat][ip]);
01169 }
01170 }

```

Here is the call graph for this function:



5.35.2.10 void module_position (met_t * met0, met_t * met1, atm_t * atm, double * dt)

Check position of air parcels.

Definition at line 1174 of file [trac.c](#).

```

01178     {
01179
01180     #ifdef _OPENACC
01181     #pragma acc data present(met0,met1,atm,dt)
01182     #pragma acc parallel loop independent gang vector
01183     #else
01184     #pragma omp parallel for default(shared)
01185     #endif
01186     for (int ip = 0; ip < atm->np; ip++)
01187         if (dt[ip] != 0) {
01188
01189             double ps, cw[3];
01190
01191             int ci[3];
01192
01193             /* Calculate modulo... */
01194             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01195             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01196
01197             /* Check latitude... */
01198             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01199                 if (atm->lat[ip] > 90) {
01200                     atm->lat[ip] = 180 - atm->lat[ip];
01201                     atm->lon[ip] += 180;
01202                 }
01203                 if (atm->lat[ip] < -90) {
01204                     atm->lat[ip] = -180 - atm->lat[ip];
01205                     atm->lon[ip] += 180;
01206                 }
01207             }
01208
01209             /* Check longitude... */

```

```

01210     while (atm->lon[ip] < -180)
01211         atm->lon[ip] += 360;
01212     while (atm->lon[ip] >= 180)
01213         atm->lon[ip] -= 360;
01214
01215     /* Check pressure... */
01216     if (atm->p[ip] < met0->p[met0->np - 1])
01217         atm->p[ip] = met0->p[met0->np - 1];
01218     else if (atm->p[ip] > 300.) {
01219         intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
01220             time[ip],
01221             atm->lon[ip], atm->lat[ip], &ps, ci, cw, 1);
01222         if (atm->p[ip] > ps)
01223             atm->p[ip] = ps;
01224     }
01225 }

```

Here is the call graph for this function:



5.35.2.11 void module_sedi (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double * *dt*)

Calculate sedimentation of air parcels.

Definition at line 1229 of file [trac.c](#).

```

01234     {
01235
01236     #ifdef _OPENACC
01237     #pragma acc data present(ctl,met0,met1,atm,dt)
01238     #pragma acc parallel loop independent gang vector
01239     #else
01240     #pragma omp parallel for default(shared)
01241     #endif
01242     for (int ip = 0; ip < atm->np; ip++)
01243         if (dt[ip] != 0) {
01244
01245             double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p, cw[3];
01246
01247             int ci[3];
01248
01249             /* Convert units... */
01250             p = 100. * atm->p[ip];
01251             r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01252             rho_p = atm->q[ctl->qnt_rho][ip];
01253
01254             /* Get temperature... */
01255             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
01256                 time[ip],
01257                 atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
01258                 1);
01259
01259             /* Density of dry air... */
01260             rho = p / (RA * T);
01261
01262             /* Dynamic viscosity of air... */
01263             eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01264
01265             /* Thermal velocity of an air molecule... */
01266             v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
01267
01268             /* Mean free path of an air molecule... */
01269             lambda = 2. * eta / (rho * v);
01270
01271             /* Knudsen number for air... */
01272             K = lambda / r_p;
01273

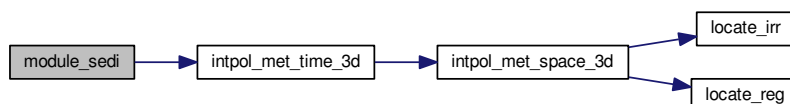
```

```

01274      /* Cunningham slip-flow correction... */
01275      G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
01276
01277      /* Sedimentation (fall) velocity... */
01278      v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01279
01280      /* Calculate pressure change... */
01281      atm->p[ip] += DZ2DP(v_p * dt[ip] / 1000., atm->p[ip]);
01282  }
01283 }

```

Here is the call graph for this function:



5.35.2.12 void module_oh_chem (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double * *dt*)

Calculate OH chemistry.

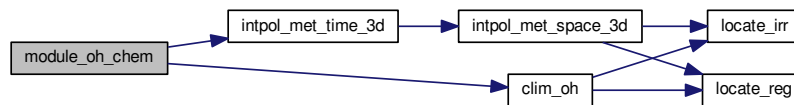
Definition at line 1287 of file [trac.c](#).

```

01292      {
01293
01294      /* Check quantity flags... */
01295      if (ctl->qnt_m < 0)
01296          ERRMSG("Module needs quantity mass!");
01297
01298      #ifdef _OPENACC
01299      #pragma acc data present(ctl,atm,dt)
01300      #pragma acc parallel loop independent gang vector
01301      #else
01302      #pragma omp parallel for default(shared)
01303      #endif
01304      for (int ip = 0; ip < atm->np; ip++)
01305          if (dt[ip] != 0) {
01306
01307              double c, k, k0, ki, M, T, cw[3];
01308
01309              int ci[3];
01310
01311              /* Get temperature... */
01312              intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01313                              atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
01314                              1);
01315
01316              /* Calculate molecular density... */
01317              M = 7.243e21 * (atm->p[ip] / P0) / T;
01318
01319              /* Calculate rate coefficient for X + OH + M -> XOH + M
(JPL Publication 15-10) ... */
01320              k0 = ctl->oh_chem[0] *
01321                  (ctl->oh_chem[1] > 0 ? pow(T / 300., -ctl->oh_chem[1]) : 1.);
01322              ki = ctl->oh_chem[2] *
01323                  (ctl->oh_chem[3] > 0 ? pow(T / 300., -ctl->oh_chem[3]) : 1.);
01324              c = log10(k0 * M / ki);
01325              k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01326
01327              /* Calculate exponential decay... */
01328              atm->q[ctl->qnt_m][ip] *=
01329                  exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->
p[ip]));
01330          }
01331      }
01332 }

```


Here is the call graph for this function:



5.35.2.13 void module_wet_deposition (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double * *dt*)

Calculate wet deposition.

Definition at line 1336 of file [trac.c](#).

```

01341     {
01342
01343     /* Check quantity flags... */
01344     if (ctl->qnt_m < 0)
01345         ERRMSG("Module needs quantity mass!");
01346
01347     #ifdef _OPENACC
01348     #pragma acc data present(ctl,atm,dt)
01349     #pragma acc parallel loop independent gang vector
01350     #else
01351     #pragma omp parallel for default(shared)
01352     #endif
01353     for (int ip = 0; ip < atm->np; ip++)
01354         if (dt[ip] != 0) {
01355
01356             double H, Is, Si, T, cl, lambda, iwc, lwc, pc, cw[3];
01357
01358             int inside, ci[3];
01359
01360             /* Check whether particle is below cloud top... */
01361             intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01362                             atm->lon[ip], atm->lat[ip], &pc, ci, cw, 1);
01363             if (!isfinite(pc) || atm->p[ip] <= pc)
01364                 continue;
01365
01366             /* Check whether particle is inside or below cloud... */
01367             intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
01368                             atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
1);
01369             intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
01370                             atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
0);
01371             inside = (iwc > 0 || lwc > 0);
01372
01373             /* Estimate precipitation rate (Pisso et al., 2019)... */
01374             intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
01375                             atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
01376             Is = pow(2. * cl, 1. / 0.36);
01377             if (Is < 0.01)
01378                 continue;
01379
01380             /* Calculate in-cloud scavenging for gases... */
01381             if (inside) {
01382
01383                 /* Get temperature... */
01384                 intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01385                             atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
0);
01386
01387                 /* Get Henry's constant (Sander, 2015)... */
01388                 H = ctl->wet_depo[2] * 101.325
01389                     * exp(ctl->wet_depo[3] * (1. / T - 1. / 298.15));
01390
01391             }
01392
01393         }
01394     }

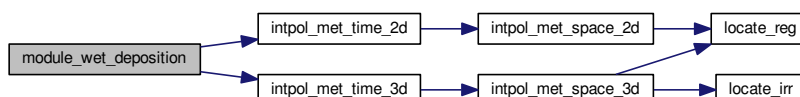
```

```

01394      /* Get scavenging coefficient (Hertel et al., 1995)... */
01395      Si = 1. / ((1. - c1) / (H * RI / P0 * T) + c1);
01396      lambda = 6.2 * Si * Is / 3.6e6;
01397  }
01398
01399      /* Calculate below-cloud scavenging for gases (Pisso et al., 2019)... */
01400      else
01401          lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01402
01403      /* Calculate exponential decay... */
01404      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
01405  }
01406 }

```

Here is the call graph for this function:



5.35.2.14 void write_output (const char * *dirname*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

Write simulation output.

Definition at line 1410 of file [trac.c](#).

```

01416      {
01417
01418      char filename[2 * LEN];
01419
01420      double r;
01421
01422      int year, mon, day, hour, min, sec, updated = 0;
01423
01424      /* Get time... */
01425      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01426
01427      /* Write atmospheric data... */
01428      if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01429          if (!updated) {
01430              #ifdef USE_NVTX
01431                  RANGE_PUSH("W atm D2H", RED);
01432              #endif
01433              #ifdef _OPENACC
01434              #pragma acc update host (atm[:1])
01435              #endif
01436              #ifdef USE_NVTX
01437                  RANGE_POP;
01438              #endif
01439              updated = 1;
01440          }
01441          #ifdef USE_NVTX
01442              RANGE_PUSH("IO", YELLOW);
01443          #endif
01444          sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01445                  dirname, ctl->atm_basename, year, mon, day, hour, min);
01446          write_atm(filename, ctl, atm, t);
01447          #ifdef USE_NVTX
01448              RANGE_POP;
01449          #endif
01450          }
01451
01452      /* Write gridded data... */
01453      if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01454          if (!updated) {
01455              #ifdef USE_NVTX
01456                  RANGE_PUSH("W grd D2H", RED);
01457              #endif
01458              #ifdef _OPENACC

```

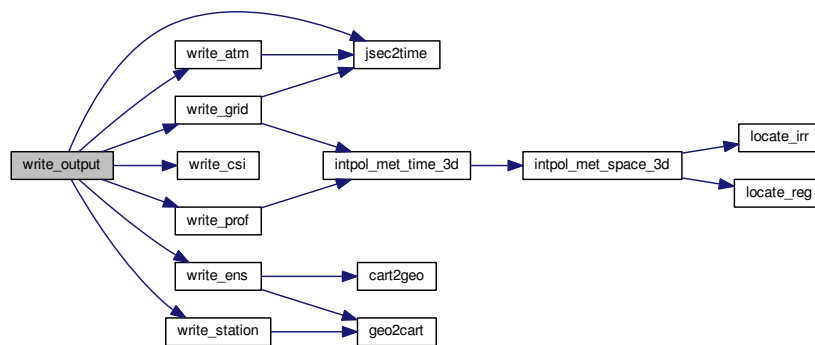
```
01459 #pragma acc update host (atm[:1])
01460 #endif
01461 #ifdef USE_NVTX
01462     RANGE_POP;
01463 #endif
01464     updated = 1;
01465 }
01466 #ifdef USE_NVTX
01467     RANGE_PUSH("IO", YELLOW);
01468 #endif
01469     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01470             dirname, ctl->grid_basename, year, mon, day, hour, min);
01471     write_grid(filename, ctl, met0, met1, atm, t);
01472 #ifdef USE_NVTX
01473     RANGE_POP;
01474 #endif
01475 }
01476
01477 /* Write CSI data... */
01478 if (ctl->csi_basename[0] != '-') {
01479     if (!updated) {
01480 #ifdef USE_NVTX
01481         RANGE_PUSH("W csi D2H", RED);
01482 #endif
01483 #ifdef _OPENACC
01484 #pragma acc update host (atm[:1])
01485 #endif
01486 #ifdef USE_NVTX
01487         RANGE_POP;
01488 #endif
01489         updated = 1;
01490     }
01491 #ifdef USE_NVTX
01492     RANGE_PUSH("IO", YELLOW);
01493 #endif
01494     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01495     write_csi(filename, ctl, atm, t);
01496 #ifdef USE_NVTX
01497     RANGE_POP;
01498 #endif
01499 }
01500
01501 /* Write ensemble data... */
01502 if (ctl->ens_basename[0] != '-') {
01503     if (!updated) {
01504 #ifdef USE_NVTX
01505         RANGE_PUSH("W csi D2H", RED);
01506 #endif
01507 #ifdef _OPENACC
01508 #pragma acc update host (atm[:1])
01509 #endif
01510 #ifdef USE_NVTX
01511         RANGE_POP;
01512 #endif
01513         updated = 1;
01514     }
01515 #ifdef USE_NVTX
01516     RANGE_PUSH("IO", YELLOW);
01517 #endif
01518     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01519     write_ens(filename, ctl, atm, t);
01520 #ifdef USE_NVTX
01521     RANGE_POP;
01522 #endif
01523 }
01524
01525 /* Write profile data... */
01526 if (ctl->prof_basename[0] != '-') {
01527     if (!updated) {
01528 #ifdef USE_NVTX
01529         RANGE_PUSH("W prof D2H", RED);
01530 #endif
01531 #ifdef _OPENACC
01532 #pragma acc update host (atm[:1])
01533 #endif
01534 #ifdef USE_NVTX
01535         RANGE_POP;
01536 #endif
01537         updated = 1;
01538     }
01539 #ifdef USE_NVTX
01540     RANGE_PUSH("IO", YELLOW);
01541 #endif
01542     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01543     write_prof(filename, ctl, met0, met1, atm, t);
01544 #ifdef USE_NVTX
01545     RANGE_POP;
```

```

01546 #endif
01547 }
01548
01549 /* Write station data... */
01550 if (ctl->stat_basename[0] != '-') {
01551     if (!updated) {
01552 #ifdef USE_NVTX
01553         RANGE_PUSH("W st D2H", RED);
01554 #endif
01555 #ifdef _OPENACC
01556 #pragma acc update host (atm[:1])
01557 #endif
01558 #ifdef USE_NVTX
01559         RANGE_POP;
01560 #endif
01561         updated = 1;
01562     }
01563 #ifdef USE_NVTX
01564     RANGE_PUSH("IO", YELLOW);
01565 #endif
01566     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01567     write_station(filename, ctl, atm, t);
01568 #ifdef USE_NVTX
01569     RANGE_POP;
01570 #endif
01571 }
01572 }

```

Here is the call graph for this function:



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

Definition at line 161 of file [trac.c](#).

```

00163     {
00164
00165     ctl_t ctl;
00166
00167     atm_t *atm;
00168
00169     cache_t *cache;
00170
00171     met_t *met0, *met1;
00172
00173     FILE *dirlist;
00174
00175     char dirname[LEN], filename[2 * LEN];
00176
00177     double *dt, *rs, t;
00178
00179     int ntask = -1, rank = 0, size = 1;
00180
00181     /* Initialize MPI... */
00182 #ifdef MPI
00183     MPI_Init(&argc, &argv);

```

```

00184 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00185 MPI_Comm_size(MPI_COMM_WORLD, &size);
00186 #endif
00187
00188 /* Initialize GPUs... */
00189 #ifndef _OPENACC
00190 #ifdef USE_NVTX
00191     RANGE_PUSH("init GPUs", GRAY);
00192 #endif
00193     acc_device_t device_type = acc_get_device_type();
00194     int num_devices = acc_get_num_devices(acc_device_nvidia);
00195     int device_num = rank % num_devices;
00196     acc_set_device_num(device_num, acc_device_nvidia);
00197     acc_init(device_type);
00198 #ifdef USE_NVTX
00199     RANGE_POP;
00200 #endif
00201 #endif
00202
00203 /* Check arguments... */
00204 if (argc < 5)
00205     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00206
00207 /* Open directory list... */
00208 if (!(dirlist = fopen(argv[1], "r")))
00209     ERRMSG("Cannot open directory list!");
00210
00211 /* Loop over directories... */
00212 while (fscanf(dirlist, "%s", dirname) != EOF) {
00213
00214     /* MPI parallelization... */
00215     if ((++ntask) % size != rank)
00216         continue;
00217
00218     /* -----
00219        Initialize model run...
00220        ----- */
00221
00222     /* Set timers... */
00223     START_TIMER(TIMER_ZERO);
00224     START_TIMER(TIMER_TOTAL);
00225     START_TIMER(TIMER_INIT);
00226
00227     /* Allocate... */
00228 #ifdef USE_NVTX
00229     RANGE_PUSH("Allocate", GRAY);
00230 #endif
00231     ALLOC(atm, atm_t, 1);
00232     ALLOC(cache, cache_t, 1);
00233     ALLOC(met0, met_t, 1);
00234     ALLOC(met1, met_t, 1);
00235     ALLOC(dt, double,
00236           NP);
00237     ALLOC(rs, double,
00238           3 * NP);
00239 #ifdef USE_NVTX
00240     RANGE_POP;
00241 #endif
00242
00243 #ifdef USE_NVTX
00244     RANGE_PUSH("Read (I/O)", GRAY);
00245 #endif
00246
00247     /* Read control parameters... */
00248     sprintf(filename, "%s/%s", dirname, argv[2]);
00249     read_ctl(filename, argc, argv, &ctl);
00250
00251     /* Read atmospheric data... */
00252     sprintf(filename, "%s/%s", dirname, argv[3]);
00253     if (!read_atm(filename, &ctl, atm))
00254         ERRMSG("Cannot open file!");
00255
00256 #ifdef USE_NVTX
00257     RANGE_POP;
00258 #endif
00259
00260     /* Copy to GPU... */
00261 #ifndef _OPENACC
00262 #ifdef USE_NVTX
00263     RANGE_PUSH("Copy to GPU", GRAY);
00264 #endif
00265     #pragma acc enter data copyin(ctl)
00266     #pragma acc enter data create(atm[:1], cache[:1], met0[:1], met1[:1], dt[:NP], rs[:3*NP])
00267     #pragma acc update device(atm[:1], cache[:1])
00268 #endif
00269 #ifdef USE_NVTX
00270     RANGE_POP;

```

```

00271 #endif
00272
00273 /* Set start time... */
00274 if (ctl.direction == 1) {
00275     ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00276     if (ctl.t_stop > 1e99)
00277         ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00278 } else {
00279     ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00280     if (ctl.t_stop > 1e99)
00281         ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00282 }
00283
00284 /* Check time interval... */
00285 if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00286     ERRMSG("Nothing to do!");
00287
00288 /* Round start time... */
00289 if (ctl.direction == 1)
00290     ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00291 else
00292     ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
dt_mod;
00293
00294 #ifdef _OPENACC
00295 #pragma acc update device(ctl)
00296 #endif
00297
00298 /* Initialize random number generator... */
00299 module_diffusion_init();
00300
00301 /* Set timers... */
00302 STOP_TIMER(TIMER_INIT);
00303
00304 /* Initialize meteorological data... */
00305 START_TIMER(TIMER_INPUT);
00306 get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
00307 if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00308     WARN("Violation of CFL criterion! Check DT_MOD!");
00309 STOP_TIMER(TIMER_INPUT);
00310
00311 /* Initialize isosurface... */
00312 START_TIMER(TIMER_ISOSURF);
00313 if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00314     module_isosurf_init(&ctl, met0, met1, atm, cache);
00315 STOP_TIMER(TIMER_ISOSURF);
00316
00317 /* -----
00318     Loop over timesteps...
00319     ----- */
00320
00321 /* Loop over timesteps... */
00322 for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
dt_mod;
00323      t += ctl.direction * ctl.dt_mod) {
00324
00325     /* Adjust length of final time step... */
00326     if (ctl.direction * (t - ctl.t_stop) > 0)
00327         t = ctl.t_stop;
00328
00329     /* Set time steps for air parcels... */
00330 #ifdef _OPENACC
00331 #ifdef USE_NVTX
00332         RANGE_PUSH("Set time steps", GRAY);
00333 #endif
00334 #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00335 #endif
00336     for (int ip = 0; ip < atm->np; ip++) {
00337         double atmtime = atm->time[ip];
00338         double tstart = ctl.t_start;
00339         double tstop = ctl.t_stop;
00340         int dir = ctl.direction;
00341         if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
&& dir * (atmtime - t) < 0))
00342             dt[ip] = t - atmtime;
00343         else
00344             dt[ip] = 0;
00345     }
00346 #ifdef USE_NVTX
00347     RANGE_POP;
00348 #endif
00349 #ifdef USE_NVTX
00350     RANGE_PUSH("Get met data", GRAY);
00351 #endif
00352 /* Get meteorological data... */
00353 START_TIMER(TIMER_INPUT);

```

```

00355     if (t != ctl.t_start)
00356         get_met(&ctl, argv[4], t, &met0, &met1);
00357     STOP_TIMER(TIMER_INPUT);
00358 #ifdef USE_NVTX
00359     RANGE_POP;
00360     RANGE_PUSH("Check init pos", GRAY);
00361 #endif
00362     /* Check initial position... */
00363     START_TIMER(TIMER_POSITION);
00364     module_position(met0, met1, atm, dt);
00365     STOP_TIMER(TIMER_POSITION);
00366 #ifdef USE_NVTX
00367     RANGE_POP;
00368     RANGE_PUSH("Advection", GRAY);
00369 #endif
00370     /* Advection... */
00371     START_TIMER(TIMER_ADVECT);
00372     module_advection(met0, met1, atm, dt);
00373     STOP_TIMER(TIMER_ADVECT);
00374 #ifdef USE_NVTX
00375     RANGE_POP;
00376     RANGE_PUSH("Turbulent diffusion", GRAY);
00377 #endif
00378     /* Turbulent diffusion... */
00379     START_TIMER(TIMER_DIFFTURB);
00380     if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00381         || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00382         module_diffusion_rng(rs, 3 * (size_t) atm->np);
00383         module_diffusion_turb(&ctl, atm, dt, rs);
00384     }
00385     STOP_TIMER(TIMER_DIFFTURB);
00386 #ifdef USE_NVTX
00387     RANGE_POP;
00388     RANGE_PUSH("Mesoscale diffusion", GRAY);
00389 #endif
00390     /* Mesoscale diffusion... */
00391     START_TIMER(TIMER_DIFFMESO);
00392     if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00393         module_diffusion_rng(rs, 3 * (size_t) atm->np);
00394         module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00395     }
00396     STOP_TIMER(TIMER_DIFFMESO);
00397 #ifdef USE_NVTX
00398     RANGE_POP;
00399     RANGE_PUSH("Sedimentation", GRAY);
00400 #endif
00401     /* Sedimentation... */
00402     START_TIMER(TIMER_SEDI);
00403     if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00404         module_sedi(&ctl, met0, met1, atm, dt);
00405     STOP_TIMER(TIMER_SEDI);
00406 #ifdef USE_NVTX
00407     RANGE_POP;
00408     RANGE_PUSH("Isosurface", GRAY);
00409 #endif
00410     /* Isosurface... */
00411     START_TIMER(TIMER_ISOSURF);
00412     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00413         module_isosurf(&ctl, met0, met1, atm, cache);
00414     STOP_TIMER(TIMER_ISOSURF);
00415 #ifdef USE_NVTX
00416     RANGE_POP;
00417     RANGE_PUSH("Check final pos", GRAY);
00418 #endif
00419     /* Check final position... */
00420     START_TIMER(TIMER_POSITION);
00421     module_position(met0, met1, atm, dt);
00422     STOP_TIMER(TIMER_POSITION);
00423 #ifdef USE_NVTX
00424     RANGE_POP;
00425     RANGE_PUSH("Interpolate met data", GRAY);
00426 #endif
00427     /* Interpolate meteorological data... */
00428     START_TIMER(TIMER_METEO);
00429     if (ctl.met_dt_out > 0
00430         && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
00431 met_dt_out) == 0))
00432         module_meteo(&ctl, met0, met1, atm);
00433     STOP_TIMER(TIMER_METEO);
00434 #ifdef USE_NVTX
00435     RANGE_POP;
00436     RANGE_PUSH("Decay of particle mass", GRAY);
00437 #endif
00438     /* Decay of particle mass... */
00439
00440

```

```

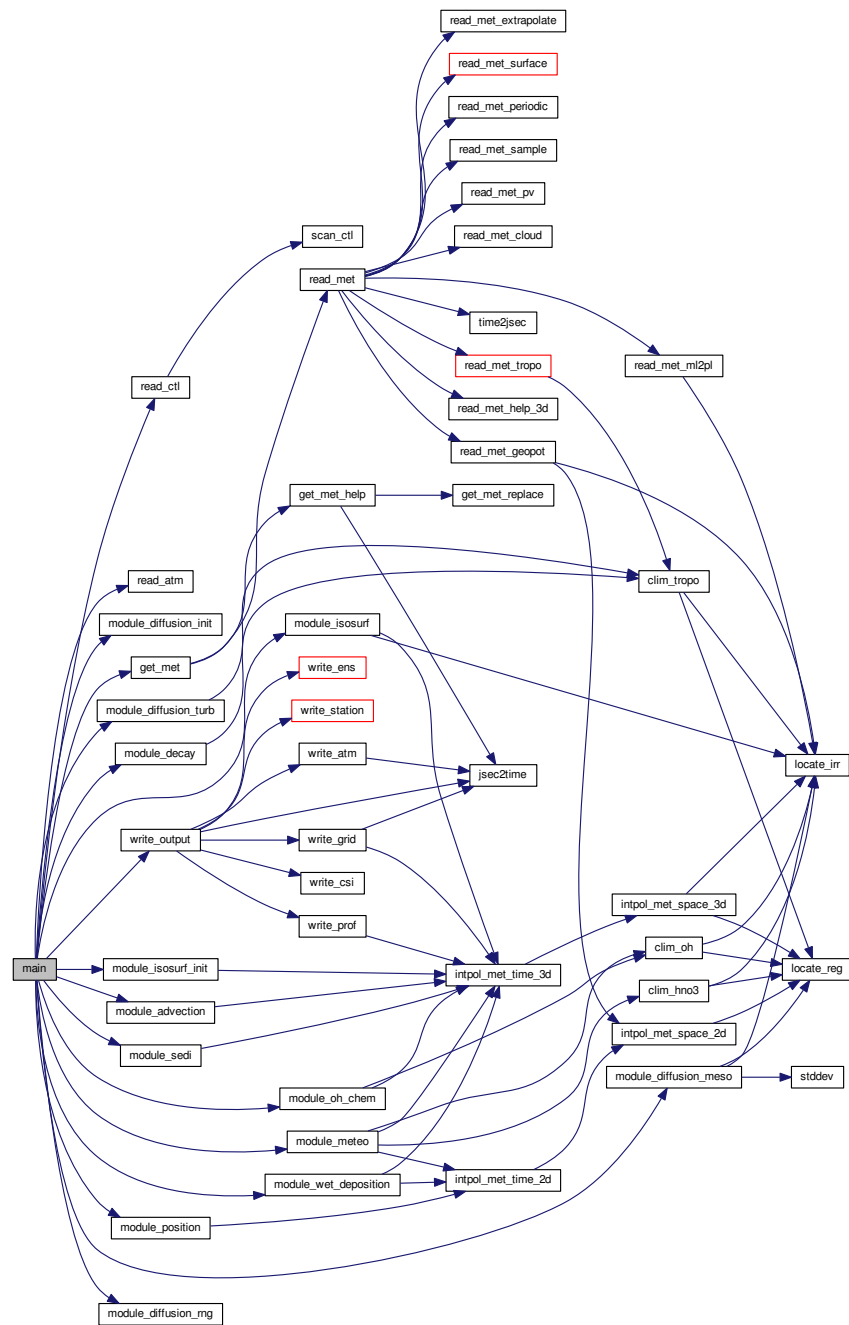
00441     START_TIMER(TIMER_DECAY);
00442     if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00443         module_decay(&ctl, atm, dt);
00444     STOP_TIMER(TIMER_DECAY);
00445 #ifdef USE_NVTX
00446     RANGE_POP;
00447     RANGE_PUSH("OH chem", GRAY);
00448 #endif
00449
00450     /* OH chemistry... */
00451     START_TIMER(TIMER_OHCHEM);
00452     if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
00453         module_oh_chem(&ctl, met0, met1, atm, dt);
00454     STOP_TIMER(TIMER_OHCHEM);
00455 #ifdef USE_NVTX
00456     RANGE_POP;
00457     RANGE_PUSH("Wet deposition", GRAY);
00458 #endif
00459
00460     /* Wet deposition... */
00461     START_TIMER(TIMER_WETDEPO);
00462     if (ctl.wet_depo[0] > 0 && ctl.wet_depo[1] > 0
00463         && ctl.wet_depo[2] > 0 && ctl.wet_depo[3] > 0)
00464         module_wet_deposition(&ctl, met0, met1, atm, dt);
00465     STOP_TIMER(TIMER_WETDEPO);
00466 #ifdef USE_NVTX
00467     RANGE_POP;
00468     RANGE_PUSH("Write output", GRAY);
00469 #endif
00470
00471     /* Write output... */
00472     START_TIMER(TIMER_OUTPUT);
00473     write_output(dirname, &ctl, met0, met1, atm, t);
00474     STOP_TIMER(TIMER_OUTPUT);
00475 #ifdef USE_NVTX
00476     RANGE_POP;
00477 #endif
00478 }
00479
00480 /* -----
00481    Finalize model run...
00482    ----- */
00483
00484 /* Report problem size... */
00485 printf("SIZE_NP = %d\n", atm->np);
00486 printf("SIZE_TASKS = %d\n", size);
00487 printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00488
00489 /* Report memory usage... */
00490 printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00491 printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00492 printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00493 printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00494                                     + 4 * NP * sizeof(double)
00495                                     + EX * EY * EP * sizeof(float)) /
00496     1024. / 1024.);
00497 printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00498                                     + EX * EY * EP * sizeof(float)
00499                                     + 4 * GX * GY * GZ * sizeof(double)
00500                                     + 2 * GX * GY * GZ * sizeof(int)
00501                                     + 2 * GX * GY * sizeof(double)
00502                                     + GX * GY * sizeof(int)) / 1024. /
00503     1024.);
00504
00505 /* Report timers... */
00506 STOP_TIMER(TIMER_ZERO);
00507 PRINT_TIMER(TIMER_INIT);
00508 PRINT_TIMER(TIMER_INPUT);
00509 PRINT_TIMER(TIMER_OUTPUT);
00510 PRINT_TIMER(TIMER_ADVECT);
00511 PRINT_TIMER(TIMER_DECAY);
00512 PRINT_TIMER(TIMER_DIFFMESO);
00513 PRINT_TIMER(TIMER_DIFFTURB);
00514 PRINT_TIMER(TIMER_ISOSURF);
00515 PRINT_TIMER(TIMER_METEO);
00516 PRINT_TIMER(TIMER_POSITION);
00517 PRINT_TIMER(TIMER_SEDI);
00518 PRINT_TIMER(TIMER_OHCHEM);
00519 PRINT_TIMER(TIMER_WETDEPO);
00520 STOP_TIMER(TIMER_TOTAL);
00521 PRINT_TIMER(TIMER_TOTAL);
00522
00523 /* Free... */
00524 #ifdef USE_NVTX
00525     RANGE_PUSH("Deallocations", GRAY);
00526 #endif
00527     free(atm);

```



```
00528     free(cache);
00529     free(met0);
00530     free(met1);
00531     free(dt);
00532     free(rs);
00533 #ifdef _OPENACC
00534 #pragma acc exit data delete(ctl,atm,cache,met0,met1,dt,rs)
00535 #endif
00536 #ifdef USE_NVTX
00537     RANGE_POP;
00538 #endif
00539 }
00540
00541 #ifdef MPI
00542     /* Finalize MPI... */
00543     MPI_Finalize();
00544 #endif
00545
00546     return EXIT_SUCCESS;
00547 }
```

Here is the call graph for this function:



5.35.3 Variable Documentation

5.35.3.1 static gsl_rng * rng

Definition at line 46 of file [trac.c](#).

5.36 trac.c

```

00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2020 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
00028 #include "mpi.h"
00029 #endif
00030
00031 #ifdef _OPENACC
00032 #include "openacc.h"
00033 #include "curand.h"
00034 #endif
00035
00036 #ifdef USE_NVTX
00037 #include "nvToolsExt.h"
00038 #include "nvtxmc.h"
00039 #endif
00040
00041 /* -----
00042   Global variables...
00043   ----- */
00044
00045 #ifdef _OPENACC
00046 curandGenerator_t rng;
00047 #else
00048 static gsl_rng *rng[NTHREADS];
00049 #endif
00050
00051 /* -----
00052   Functions...
00053   ----- */
00054
00056 void module_advection(
00057     met_t * met0,
00058     met_t * met1,
00059     atm_t * atm,
00060     double *dt);
00061
00063 void module_decay(
00064     ctl_t * ctl,
00065     atm_t * atm,
00066     double *dt);
00067
00069 void module_diffusion_init(
00070     void);
00071
00073 void module_diffusion_meso(
00074     ctl_t * ctl,
00075     met_t * met0,
00076     met_t * met1,
00077     atm_t * atm,
00078     cache_t * cache,
00079     double *dt,
00080     double *rs);
00081
00083 void module_diffusion_rng(
00084     double *rs,
00085     size_t n);
00086
00088 void module_diffusion_turb(
00089     ctl_t * ctl,
00090     atm_t * atm,
00091     double *dt,
00092     double *rs);
00093
00095 void module_isosurf_init(
00096     ctl_t * ctl,

```

```

00097     met_t * met0,
00098     met_t * met1,
00099     atm_t * atm,
00100     cache_t * cache);
00101
00103 void module_isosurf(
00104     ctl_t * ctl,
00105     met_t * met0,
00106     met_t * met1,
00107     atm_t * atm,
00108     cache_t * cache);
00109
00111 void module_meteo(
00112     ctl_t * ctl,
00113     met_t * met0,
00114     met_t * met1,
00115     atm_t * atm);
00116
00118 void module_position(
00119     met_t * met0,
00120     met_t * met1,
00121     atm_t * atm,
00122     double *dt);
00123
00125 void module_sedi(
00126     ctl_t * ctl,
00127     met_t * met0,
00128     met_t * met1,
00129     atm_t * atm,
00130     double *dt);
00131
00133 void module_oh_chem(
00134     ctl_t * ctl,
00135     met_t * met0,
00136     met_t * met1,
00137     atm_t * atm,
00138     double *dt);
00139
00141 void module_wet_deposition(
00142     ctl_t * ctl,
00143     met_t * met0,
00144     met_t * met1,
00145     atm_t * atm,
00146     double *dt);
00147
00149 void write_output(
00150     const char *dirname,
00151     ctl_t * ctl,
00152     met_t * met0,
00153     met_t * met1,
00154     atm_t * atm,
00155     double t);
00156
00157 /* -----
00158     Main...
00159     ----- */
00160
00161 int main(
00162     int argc,
00163     char *argv[]) {
00164
00165     ctl_t ctl;
00166
00167     atm_t *atm;
00168
00169     cache_t *cache;
00170
00171     met_t *met0, *met1;
00172
00173     FILE *dirlist;
00174
00175     char dirname[LEN], filename[2 * LEN];
00176
00177     double *dt, *rs, t;
00178
00179     int ntask = -1, rank = 0, size = 1;
00180
00181     /* Initialize MPI... */
00182     #ifdef MPI
00183         MPI_Init(&argc, &argv);
00184         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00185         MPI_Comm_size(MPI_COMM_WORLD, &size);
00186     #endif
00187
00188     /* Initialize GPUs... */
00189     #ifdef _OPENACC
00190     #ifdef USE_NVTX

```

```

00191 RANGE_PUSH("init GPUs", GRAY);
00192 #endif
00193 acc_device_t device_type = acc_get_device_type();
00194 int num_devices = acc_get_num_devices(acc_device_nvidia);
00195 int device_num = rank % num_devices;
00196 acc_set_device_num(device_num, acc_device_nvidia);
00197 acc_init(device_type);
00198 #ifdef USE_NVTX
00199 RANGE_POP;
00200 #endif
00201 #endif
00202
00203 /* Check arguments... */
00204 if (argc < 5)
00205     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00206
00207 /* Open directory list... */
00208 if (!(dirlist = fopen(argv[1], "r")))
00209     ERRMSG("Cannot open directory list!");
00210
00211 /* Loop over directories... */
00212 while (fscanf(dirlist, "%s", dirname) != EOF) {
00213
00214     /* MPI parallelization... */
00215     if ((++ntask) % size != rank)
00216         continue;
00217
00218     /* -----
00219        Initialize model run...
00220        ----- */
00221
00222     /* Set timers... */
00223     START_TIMER(TIMER_ZERO);
00224     START_TIMER(TIMER_TOTAL);
00225     START_TIMER(TIMER_INIT);
00226
00227     /* Allocate... */
00228 #ifdef USE_NVTX
00229     RANGE_PUSH("Allocate", GRAY);
00230 #endif
00231     ALLOC(atm, atm_t, 1);
00232     ALLOC(cache, cache_t, 1);
00233     ALLOC(met0, met_t, 1);
00234     ALLOC(met1, met_t, 1);
00235     ALLOC(dt, double,
00236           NP);
00237     ALLOC(rs, double,
00238           3 * NP);
00239 #ifdef USE_NVTX
00240     RANGE_POP;
00241 #endif
00242
00243 #ifdef USE_NVTX
00244     RANGE_PUSH("Read (I/O)", GRAY);
00245 #endif
00246
00247     /* Read control parameters... */
00248     sprintf(filename, "%s/%s", dirname, argv[2]);
00249     read_ctl(filename, argc, argv, &ctl);
00250
00251     /* Read atmospheric data... */
00252     sprintf(filename, "%s/%s", dirname, argv[3]);
00253     if (!read_atm(filename, &ctl, atm))
00254         ERRMSG("Cannot open file!");
00255
00256 #ifdef USE_NVTX
00257     RANGE_POP;
00258 #endif
00259
00260     /* Copy to GPU... */
00261 #ifdef _OPENACC
00262 #ifdef USE_NVTX
00263     RANGE_PUSH("Copy to GPU", GRAY);
00264 #endif
00265     #pragma acc enter data copyin(ctl)
00266     #pragma acc enter data create(atm[:1], cache[:1], met0[:1], met1[:1], dt[:NP], rs[:3*NP])
00267     #pragma acc update device(atm[:1], cache[:1])
00268 #endif
00269 #ifdef USE_NVTX
00270     RANGE_POP;
00271 #endif
00272
00273     /* Set start time... */
00274     if (ctl.direction == 1) {
00275         ctl.t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00276         if (ctl.t_stop > 1e99)
00277             ctl.t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);

```

```

00278     } else {
00279         ctl.t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00280         if (ctl.t_stop > 1e99)
00281             ctl.t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00282     }
00283
00284     /* Check time interval... */
00285     if (ctl.direction * (ctl.t_stop - ctl.t_start) <= 0)
00286         ERRMSG("Nothing to do!");
00287
00288     /* Round start time... */
00289     if (ctl.direction == 1)
00290         ctl.t_start = floor(ctl.t_start / ctl.dt_mod) * ctl.
00291         dt_mod;
00292     else
00293         ctl.t_start = ceil(ctl.t_start / ctl.dt_mod) * ctl.
00294         dt_mod;
00295
00296 #ifdef _OPENACC
00297 #pragma acc update device(ctl)
00298 #endif
00299
00300     /* Initialize random number generator... */
00301     module_diffusion_init();
00302
00303     /* Set timers... */
00304     STOP_TIMER(TIMER_INIT);
00305
00306     /* Initialize meteorological data... */
00307     START_TIMER(TIMER_INPUT);
00308     get_met(&ctl, argv[4], ctl.t_start, &met0, &met1);
00309     if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00310         WARN("Violation of CFL criterion! Check DT_MOD!");
00311     STOP_TIMER(TIMER_INPUT);
00312
00313     /* Initialize isosurface... */
00314     START_TIMER(TIMER_ISOSURF);
00315     if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00316         module_isosurf_init(&ctl, met0, met1, atm, cache);
00317     STOP_TIMER(TIMER_ISOSURF);
00318
00319     /* -----
00320     Loop over timesteps...
00321     ----- */
00322
00323     /* Loop over timesteps... */
00324     for (t = ctl.t_start; ctl.direction * (t - ctl.t_stop) < ctl.
00325         dt_mod;
00326         t += ctl.direction * ctl.dt_mod) {
00327
00328         /* Adjust length of final time step... */
00329         if (ctl.direction * (t - ctl.t_stop) > 0)
00330             t = ctl.t_stop;
00331
00332         /* Set time steps for air parcels... */
00333 #ifdef _OPENACC
00334 #ifdef USE_NVTX
00335         RANGE_PUSH("Set time steps", GRAY);
00336 #endif
00337 #pragma acc parallel loop independent gang vector present(ctl,atm,atm->time,dt)
00338 #endif
00339         for (int ip = 0; ip < atm->np; ip++) {
00340             double atmtime = atm->time[ip];
00341             double tstart = ctl.t_start;
00342             double tstop = ctl.t_stop;
00343             int dir = ctl.direction;
00344             if ((dir * (atmtime - tstart) >= 0 && dir * (atmtime - tstop) <= 0
00345                 && dir * (atmtime - t) < 0))
00346                 dt[ip] = t - atmtime;
00347             else
00348                 dt[ip] = 0;
00349         }
00350 #ifdef USE_NVTX
00351         RANGE_POP;
00352 #endif
00353 #ifdef USE_NVTX
00354         RANGE_PUSH("Get met data", GRAY);
00355 #endif
00356         /* Get meteorological data... */
00357         START_TIMER(TIMER_INPUT);
00358         if (t != ctl.t_start)
00359             get_met(&ctl, argv[4], t, &met0, &met1);
00360         STOP_TIMER(TIMER_INPUT);
00361 #ifdef USE_NVTX
00362         RANGE_POP;
00363         RANGE_PUSH("Check init pos", GRAY);
00364 #endif
00365 #endif

```

```

00362      /* Check initial position... */
00363      START_TIMER(TIMER_POSITION);
00364      module_position(met0, met1, atm, dt);
00365      STOP_TIMER(TIMER_POSITION);
00366 #ifdef USE_NVTX
00367      RANGE_POP;
00368      RANGE_PUSH("Advection", GRAY);
00369 #endif
00370      /* Advection... */
00371      START_TIMER(TIMER_ADVECT);
00372      module_advection(met0, met1, atm, dt);
00373      STOP_TIMER(TIMER_ADVECT);
00374 #ifdef USE_NVTX
00375      RANGE_POP;
00376      RANGE_PUSH("Turbulent diffusion", GRAY);
00377 #endif
00378      /* Turbulent diffusion... */
00379      START_TIMER(TIMER_DIFFTURB);
00380      if (ctl.turb_dx_trop > 0 || ctl.turb_dz_trop > 0
00381          || ctl.turb_dx_strat > 0 || ctl.turb_dz_strat > 0) {
00382          module_diffusion_rng(rs, 3 * (size_t) atm->np);
00383          module_diffusion_turb(&ctl, atm, dt, rs);
00384      }
00385      STOP_TIMER(TIMER_DIFFTURB);
00386 #ifdef USE_NVTX
00387      RANGE_POP;
00388      RANGE_PUSH("Mesoscale diffusion", GRAY);
00389 #endif
00390      /* Mesoscale diffusion... */
00391      START_TIMER(TIMER_DIFFMESO);
00392      if (ctl.turb_mesox > 0 || ctl.turb_mesoz > 0) {
00393          module_diffusion_rng(rs, 3 * (size_t) atm->np);
00394          module_diffusion_meso(&ctl, met0, met1, atm, cache, dt, rs);
00395      }
00396      STOP_TIMER(TIMER_DIFFMESO);
00397 #ifdef USE_NVTX
00398      RANGE_POP;
00399      RANGE_PUSH("Sedimentation", GRAY);
00400 #endif
00401      /* Sedimentation... */
00402      START_TIMER(TIMER_SEDI);
00403      if (ctl.qnt_r >= 0 && ctl.qnt_rho >= 0)
00404          module_sedi(&ctl, met0, met1, atm, dt);
00405      STOP_TIMER(TIMER_SEDI);
00406 #ifdef USE_NVTX
00407      RANGE_POP;
00408      RANGE_PUSH("Isosurface", GRAY);
00409 #endif
00410
00411      /* Isosurface... */
00412      START_TIMER(TIMER_ISOSURF);
00413      if (ctl.isosurf >= 1 && ctl.isosurf <= 4)
00414          module_isosurf(&ctl, met0, met1, atm, cache);
00415      STOP_TIMER(TIMER_ISOSURF);
00416 #ifdef USE_NVTX
00417      RANGE_POP;
00418      RANGE_PUSH("Check final pos", GRAY);
00419 #endif
00420
00421      /* Check final position... */
00422      START_TIMER(TIMER_POSITION);
00423      module_position(met0, met1, atm, dt);
00424      STOP_TIMER(TIMER_POSITION);
00425 #ifdef USE_NVTX
00426      RANGE_POP;
00427      RANGE_PUSH("Interpolate met data", GRAY);
00428 #endif
00429      /* Interpolate meteorological data... */
00430      START_TIMER(TIMER_METEO);
00431      if (ctl.met_dt_out > 0
00432          && (ctl.met_dt_out < ctl.dt_mod || fmod(t, ctl.
00433 met_dt_out) == 0))
00434          module_meteo(&ctl, met0, met1, atm);
00435      STOP_TIMER(TIMER_METEO);
00436 #ifdef USE_NVTX
00437      RANGE_POP;
00438      RANGE_PUSH("Decay of particle mass", GRAY);
00439 #endif
00440      /* Decay of particle mass... */
00441      START_TIMER(TIMER_DECAY);
00442      if (ctl.tdec_trop > 0 && ctl.tdec_strat > 0)
00443          module_decay(&ctl, atm, dt);
00444      STOP_TIMER(TIMER_DECAY);
00445 #ifdef USE_NVTX
00446      RANGE_POP;
00447      RANGE_PUSH("OH chem", GRAY);

```

```

00448 #endif
00449
00450 /* OH chemistry... */
00451 START_TIMER(TIMER_OHCHEM);
00452 if (ctl.oh_chem[0] > 0 && ctl.oh_chem[2] > 0)
00453     module_oh_chem(&ctl, met0, met1, atm, dt);
00454 STOP_TIMER(TIMER_OHCHEM);
00455 #ifdef USE_NVTX
00456 RANGE_POP;
00457 RANGE_PUSH("Wet deposition", GRAY);
00458 #endif
00459
00460 /* Wet deposition... */
00461 START_TIMER(TIMER_WETDEPO);
00462 if (ctl.wet_depo[0] > 0 && ctl.wet_depo[1] > 0
00463     && ctl.wet_depo[2] > 0 && ctl.wet_depo[3] > 0)
00464     module_wet_deposition(&ctl, met0, met1, atm, dt);
00465 STOP_TIMER(TIMER_WETDEPO);
00466 #ifdef USE_NVTX
00467 RANGE_POP;
00468 RANGE_PUSH("Write output", GRAY);
00469 #endif
00470
00471 /* Write output... */
00472 START_TIMER(TIMER_OUTPUT);
00473 write_output(dirname, &ctl, met0, met1, atm, t);
00474 STOP_TIMER(TIMER_OUTPUT);
00475 #ifdef USE_NVTX
00476 RANGE_POP;
00477 #endif
00478 }
00479
00480 /* -----
00481 Finalize model run...
00482 ----- */
00483
00484 /* Report problem size... */
00485 printf("SIZE_NP = %d\n", atm->np);
00486 printf("SIZE_TASKS = %d\n", size);
00487 printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00488
00489 /* Report memory usage... */
00490 printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00491 printf("MEMORY_CACHE = %g MByte\n", sizeof(cache_t) / 1024. / 1024.);
00492 printf("MEMORY_METEO = %g MByte\n", 2 * sizeof(met_t) / 1024. / 1024.);
00493 printf("MEMORY_DYNAMIC = %g MByte\n", (sizeof(met_t)
00494     + 4 * NP * sizeof(double)
00495     + EX * EY * EP * sizeof(float)) /
00496     1024. / 1024.);
00497 printf("MEMORY_STATIC = %g MByte\n", (EX * EY * sizeof(double)
00498     + EX * EY * EP * sizeof(float)
00499     + 4 * GX * GY * GZ * sizeof(double)
00500     + 2 * GX * GY * GZ * sizeof(int)
00501     + 2 * GX * GY * sizeof(double)
00502     + GX * GY * sizeof(int)) / 1024. /
00503     1024.);
00504
00505 /* Report timers... */
00506 STOP_TIMER(TIMER_ZERO);
00507 PRINT_TIMER(TIMER_INIT);
00508 PRINT_TIMER(TIMER_INPUT);
00509 PRINT_TIMER(TIMER_OUTPUT);
00510 PRINT_TIMER(TIMER_ADVECT);
00511 PRINT_TIMER(TIMER_DECAY);
00512 PRINT_TIMER(TIMER_DIFFMESO);
00513 PRINT_TIMER(TIMER_DIFFTURB);
00514 PRINT_TIMER(TIMER_ISOSURF);
00515 PRINT_TIMER(TIMER_METEO);
00516 PRINT_TIMER(TIMER_POSITION);
00517 PRINT_TIMER(TIMER_SEDI);
00518 PRINT_TIMER(TIMER_OHCHEM);
00519 PRINT_TIMER(TIMER_WETDEPO);
00520 STOP_TIMER(TIMER_TOTAL);
00521 PRINT_TIMER(TIMER_TOTAL);
00522
00523 /* Free... */
00524 #ifdef USE_NVTX
00525 RANGE_PUSH("Deallocations", GRAY);
00526 #endif
00527 free(atm);
00528 free(cache);
00529 free(met0);
00530 free(met1);
00531 free(dt);
00532 free(rs);
00533 #ifdef _OPENACC
00534 #pragma acc exit data delete(ctl, atm, cache, met0, met1, dt, rs)

```



```

00535 #endif
00536 #ifdef USE_NVTX
00537     RANGE_POP;
00538 #endif
00539 }
00540
00541 #ifdef MPI
00542     /* Finalize MPI... */
00543     MPI_Finalize();
00544 #endif
00545
00546     return EXIT_SUCCESS;
00547 }
00548
00549 /*****
00550
00551 void module_advection(
00552     met_t * met0,
00553     met_t * met1,
00554     atm_t * atm,
00555     double *dt) {
00556
00557     #ifdef _OPENACC
00558     #pragma acc data present(met0,met1,atm,dt)
00559     #pragma acc parallel loop independent gang vector
00560     #else
00561     #pragma omp parallel for default(shared)
00562     #endif
00563     for (int ip = 0; ip < atm->np; ip++)
00564         if (dt[ip] != 0) {
00565
00566             int ci[3] = { 0 };
00567
00568             double dtm = 0.0, v[3] = { 0.0 }, xm[3] = {
00569                 0.0;
00570             double cw[3] = { 0.0 };
00571
00572             /* Interpolate meteorological data... */
00573             intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
00574                             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[0], ci,
00575                             cw, 1);
00576             intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
00577                             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[1], ci,
00578                             cw, 0);
00579             intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
00580                             atm->p[ip], atm->lon[ip], atm->lat[ip], &v[2], ci,
00581                             cw, 0);
00582
00583             /* Get position of the mid point... */
00584             dtm = atm->time[ip] + 0.5 * dt[ip];
00585             xm[0] =
00586                 atm->lon[ip] + DX2DEG(0.5 * dt[ip] * v[0] / 1000., atm->lat[ip]);
00587             xm[1] = atm->lat[ip] + DY2DEG(0.5 * dt[ip] * v[1] / 1000.);
00588             xm[2] = atm->p[ip] + 0.5 * dt[ip] * v[2];
00589
00590             /* Interpolate meteorological data for mid point... */
00591             intpol_met_time_3d(met0, met0->u, met1, met1->u, dtm, xm[2], xm[0],
00592                             xm[1], &v[0], ci, cw, 1);
00593             intpol_met_time_3d(met0, met0->v, met1, met1->v, dtm, xm[2], xm[0],
00594                             xm[1], &v[1], ci, cw, 0);
00595             intpol_met_time_3d(met0, met0->w, met1, met1->w, dtm, xm[2], xm[0],
00596                             xm[1], &v[2], ci, cw, 0);
00597
00598             /* Save new position... */
00599             atm->time[ip] += dt[ip];
00600             atm->lon[ip] += DX2DEG(dtm * v[0] / 1000., xm[1]);
00601             atm->lat[ip] += DY2DEG(dtm * v[1] / 1000.);
00602             atm->p[ip] += dt[ip] * v[2];
00603         }
00604 }
00605
00606 /*****
00607
00608 void module_decay(
00609     ctl_t * ctl,
00610     atm_t * atm,
00611     double *dt) {
00612
00613     /* Check quantity flags... */
00614     if (ctl->qnt_m < 0)
00615         ERRMSG("Module needs quantity mass!");
00616
00617     #ifdef _OPENACC
00618     #pragma acc data present(ctl,atm,dt)

```

```

00619 #pragma acc parallel loop independent gang vector
00620 #else
00621 #pragma omp parallel for default(shared)
00622 #endif
00623     for (int ip = 0; ip < atm->np; ip++)
00624         if (dt[ip] != 0) {
00625             double p0, p1, pt, tdec, w;
00626
00627             /* Get tropopause pressure... */
00628             pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00629
00630             /* Get weighting factor... */
00631             p1 = pt * 0.866877899;
00632             p0 = pt / 0.866877899;
00633             if (atm->p[ip] > p0)
00634                 w = 1;
00635             else if (atm->p[ip] < p1)
00636                 w = 0;
00637             else
00638                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00639
00640             /* Set lifetime... */
00641             tdec = w * ctl->tdec_trop + (1 - w) * ctl->tdec_strat;
00642
00643             /* Calculate exponential decay... */
00644             atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] / tdec);
00645         }
00646     }
00647 }
00648
00649 /*****
00650
00651 void module_diffusion_init(
00652     void) {
00653     /* Initialize random number generator... */
00654     #ifdef _OPENACC
00655     if (curandCreateGenerator(&rng, CURAND_RNG_PSEUDO_DEFAULT)
00656         != CURAND_STATUS_SUCCESS)
00657         ERRMSG("Cannot create random number generator!");
00658     if (curandSetStream(rng, (cudaStream_t) acc_get_cuda_stream(acc_async_sync))
00659         != CURAND_STATUS_SUCCESS)
00660         ERRMSG("Cannot set stream for random number generator!");
00661     #else
00662     gsl_rng_env_setup();
00663     if (omp_get_max_threads() > NTHREADS)
00664         ERRMSG("Too many threads!");
00665     for (int i = 0; i < NTHREADS; i++) {
00666         rng[i] = gsl_rng_alloc(gsl_rng_default);
00667         gsl_rng_set(rng[i], gsl_rng_default_seed + (long unsigned) i);
00668     }
00669     #endif
00670 }
00671
00672 void module_diffusion_meso(
00673     ctl_t * ctl,
00674     met_t * met0,
00675     met_t * met1,
00676     atm_t * atm,
00677     cache_t * cache,
00678     double *dt,
00679     double *rs) {
00680     #ifdef _OPENACC
00681     #pragma acc data present (ctl,met0,met1,atm,cache,dt,rs)
00682     #pragma acc parallel loop independent gang vector
00683     #else
00684     #pragma omp parallel for default(shared)
00685     #endif
00686     for (int ip = 0; ip < atm->np; ip++)
00687         if (dt[ip] != 0) {
00688             double u[16], v[16], w[16];
00689
00690             /* Get indices... */
00691             int ix = locate_reg(met0->lon, met0->nx, atm->lon[ip]);
00692             int iy = locate_reg(met0->lat, met0->ny, atm->lat[ip]);
00693             int iz = locate_irr(met0->p, met0->np, atm->p[ip]);
00694
00695             /* Caching of wind standard deviations... */
00696             if (cache->tsig[ix][iy][iz] != met0->time) {

```

```

00706
00707     /* Collect local wind data... */
00708     u[0] = met0->u[ix][iy][iz];
00709     u[1] = met0->u[ix + 1][iy][iz];
00710     u[2] = met0->u[ix][iy + 1][iz];
00711     u[3] = met0->u[ix + 1][iy + 1][iz];
00712     u[4] = met0->u[ix][iy][iz + 1];
00713     u[5] = met0->u[ix + 1][iy][iz + 1];
00714     u[6] = met0->u[ix][iy + 1][iz + 1];
00715     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00716
00717     v[0] = met0->v[ix][iy][iz];
00718     v[1] = met0->v[ix + 1][iy][iz];
00719     v[2] = met0->v[ix][iy + 1][iz];
00720     v[3] = met0->v[ix + 1][iy + 1][iz];
00721     v[4] = met0->v[ix][iy][iz + 1];
00722     v[5] = met0->v[ix + 1][iy][iz + 1];
00723     v[6] = met0->v[ix][iy + 1][iz + 1];
00724     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00725
00726     w[0] = met0->w[ix][iy][iz];
00727     w[1] = met0->w[ix + 1][iy][iz];
00728     w[2] = met0->w[ix][iy + 1][iz];
00729     w[3] = met0->w[ix + 1][iy + 1][iz];
00730     w[4] = met0->w[ix][iy][iz + 1];
00731     w[5] = met0->w[ix + 1][iy][iz + 1];
00732     w[6] = met0->w[ix][iy + 1][iz + 1];
00733     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00734
00735     /* Collect local wind data... */
00736     u[8] = met1->u[ix][iy][iz];
00737     u[9] = met1->u[ix + 1][iy][iz];
00738     u[10] = met1->u[ix][iy + 1][iz];
00739     u[11] = met1->u[ix + 1][iy + 1][iz];
00740     u[12] = met1->u[ix][iy][iz + 1];
00741     u[13] = met1->u[ix + 1][iy][iz + 1];
00742     u[14] = met1->u[ix][iy + 1][iz + 1];
00743     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00744
00745     v[8] = met1->v[ix][iy][iz];
00746     v[9] = met1->v[ix + 1][iy][iz];
00747     v[10] = met1->v[ix][iy + 1][iz];
00748     v[11] = met1->v[ix + 1][iy + 1][iz];
00749     v[12] = met1->v[ix][iy][iz + 1];
00750     v[13] = met1->v[ix + 1][iy][iz + 1];
00751     v[14] = met1->v[ix][iy + 1][iz + 1];
00752     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00753
00754     w[8] = met1->w[ix][iy][iz];
00755     w[9] = met1->w[ix + 1][iy][iz];
00756     w[10] = met1->w[ix][iy + 1][iz];
00757     w[11] = met1->w[ix + 1][iy + 1][iz];
00758     w[12] = met1->w[ix][iy][iz + 1];
00759     w[13] = met1->w[ix + 1][iy][iz + 1];
00760     w[14] = met1->w[ix][iy + 1][iz + 1];
00761     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00762
00763     /* Get standard deviations of local wind data... */
00764     cache->usig[ix][iy][iz] = (float) stddev(u, 16);
00765     cache->vsig[ix][iy][iz] = (float) stddev(v, 16);
00766     cache->wsig[ix][iy][iz] = (float) stddev(w, 16);
00767     cache->tsig[ix][iy][iz] = met0->time;
00768 }
00769
00770 /* Set temporal correlations for mesoscale fluctuations... */
00771 double r = 1 - 2 * fabs(dt[ip]) / ctl->dt_met;
00772 double r2 = sqrt(1 - r * r);
00773
00774 /* Calculate horizontal mesoscale wind fluctuations... */
00775 if (ctl->turb_mesox > 0) {
00776     cache->up[ip] = (float)
00777         (r * cache->up[ip]
00778          + r2 * rs[3 * ip] * ctl->turb_mesox * cache->usig[ix][iy][iz]);
00779     atm->lon[ip] += DX2DEG(cache->up[ip] * dt[ip] / 1000., atm->lat[ip]);
00780
00781     cache->vp[ip] = (float)
00782         (r * cache->vp[ip]
00783          + r2 * rs[3 * ip + 1] * ctl->turb_mesox * cache->vsig[ix][iy][iz]);
00784     atm->lat[ip] += DY2DEG(cache->vp[ip] * dt[ip] / 1000.);
00785 }
00786
00787 /* Calculate vertical mesoscale wind fluctuations... */
00788 if (ctl->turb_mesoz > 0) {
00789     cache->wp[ip] = (float)
00790         (r * cache->wp[ip]
00791          + r2 * rs[3 * ip + 2] * ctl->turb_mesoz * cache->wsig[ix][iy][iz]);
00792     atm->p[ip] += cache->wp[ip] * dt[ip];

```

```

00793     }
00794     }
00795 }
00796
00797 /*****
00798
00799 void module_diffusion_rng(
00800     double *rs,
00801     size_t n) {
00802
00803 #ifdef _OPENACC
00804
00805 #pragma acc host_data use_device(rs)
00806     {
00807         if (curandGenerateNormalDouble(rng, rs, n, 0.0, 1.0)
00808             != CURAND_STATUS_SUCCESS)
00809             ERRMSG("Cannot create random numbers!");
00810     }
00811 #else
00812
00813 #pragma omp parallel for default(shared)
00814     for (size_t i = 0; i < n; ++i)
00815         rs[i] = gsl_ran_gaussian_ziggurat(rng[omp_get_thread_num()], 1.0);
00816 #endif
00817 }
00818
00819
00820 }
00821
00822 /*****
00823
00824 void module_diffusion_turb(
00825     ctl_t * ctl,
00826     atm_t * atm,
00827     double *dt,
00828     double *rs) {
00829
00830 #ifdef _OPENACC
00831 #pragma acc data present(ctl,atm,dt,rs)
00832 #pragma acc parallel loop independent gang vector
00833 #else
00834 #pragma omp parallel for default(shared)
00835 #endif
00836     for (int ip = 0; ip < atm->np; ip++)
00837         if (dt[ip] != 0) {
00838
00839             double w;
00840
00841             /* Get tropopause pressure... */
00842             double pt = clim_tropo(atm->time[ip], atm->lat[ip]);
00843
00844             /* Get weighting factor... */
00845             double p1 = pt * 0.866877899;
00846             double p0 = pt / 0.866877899;
00847             if (atm->p[ip] > p0)
00848                 w = 1;
00849             else if (atm->p[ip] < p1)
00850                 w = 0;
00851             else
00852                 w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00853
00854             /* Set diffusivity... */
00855             double dx = w * ctl->turb_dx_trop + (1 - w) * ctl->
turb_dx_strat;
00856             double dz = w * ctl->turb_dz_trop + (1 - w) * ctl->
turb_dz_strat;
00857
00858             /* Horizontal turbulent diffusion... */
00859             if (dx > 0) {
00860                 double sigma = sqrt(2.0 * dx * fabs(dt[ip]));
00861                 atm->lon[ip] += DX2DEG(rs[3 * ip] * sigma / 1000., atm->lat[ip]);
00862                 atm->lat[ip] += DY2DEG(rs[3 * ip + 1] * sigma / 1000.);
00863             }
00864
00865             /* Vertical turbulent diffusion... */
00866             if (dz > 0) {
00867                 double sigma = sqrt(2.0 * dz * fabs(dt[ip]));
00868                 atm->p[ip]
00869                     += DZ2DP(rs[3 * ip + 2] * sigma / 1000., atm->p[ip]);
00870             }
00871         }
00872 }
00873
00874 /*****
00875
00876 void module_isosurf_init(
00877     ctl_t * ctl,

```

```

00878     met_t * met0,
00879     met_t * met1,
00880     atm_t * atm,
00881     cache_t * cache) {
00882
00883     FILE *in;
00884
00885     char line[LEN];
00886
00887     double t, cw[3];
00888
00889     int ci[3];
00890
00891     /* Save pressure... */
00892     if (ctl->isosurf == 1)
00893         for (int ip = 0; ip < atm->np; ip++)
00894             cache->iso_var[ip] = atm->p[ip];
00895
00896     /* Save density... */
00897     else if (ctl->isosurf == 2)
00898         for (int ip = 0; ip < atm->np; ip++) {
00899             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00900                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00901                             1);
00902             cache->iso_var[ip] = atm->p[ip] / t;
00903         }
00904
00905     /* Save potential temperature... */
00906     else if (ctl->isosurf == 3)
00907         for (int ip = 0; ip < atm->np; ip++) {
00908             intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00909                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00910                             1);
00911             cache->iso_var[ip] = THETA(atm->p[ip], t);
00912         }
00913
00914     /* Read balloon pressure data... */
00915     else if (ctl->isosurf == 4) {
00916
00917         /* Write info... */
00918         printf("Read balloon pressure data: %s\n", ctl->balloon);
00919
00920         /* Open file... */
00921         if (!(in = fopen(ctl->balloon, "r")))
00922             ERRMSG("Cannot open file!");
00923
00924         /* Read pressure time series... */
00925         while (fgets(line, LEN, in))
00926             if (sscanf(line, "%lg %lg", &(cache->iso_ts[cache->iso_n]),
00927                     &(cache->iso_ps[cache->iso_n])) == 2)
00928                 if (++cache->iso_n > NP)
00929                     ERRMSG("Too many data points!");
00930
00931         /* Check number of points... */
00932         if (cache->iso_n < 1)
00933             ERRMSG("Could not read any data!");
00934
00935         /* Close file... */
00936         fclose(in);
00937     }
00938 }
00939
00940 /*****
00941
00942 void module_isosurf(
00943     ctl_t * ctl,
00944     met_t * met0,
00945     met_t * met1,
00946     atm_t * atm,
00947     cache_t * cache) {
00948
00949 #ifdef _OPENACC
00950 #pragma acc data present(ctl,met0,met1,atm,cache)
00951 #pragma acc parallel loop independent gang vector
00952 #else
00953 #pragma omp parallel for default(shared)
00954 #endif
00955     for (int ip = 0; ip < atm->np; ip++) {
00956
00957         double t, cw[3];
00958
00959         int ci[3];
00960
00961         /* Restore pressure... */
00962         if (ctl->isosurf == 1)

```

```

00963     atm->p[ip] = cache->iso_var[ip];
00964
00965     /* Restore density... */
00966     else if (ctl->isosurf == 2) {
00967         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00968                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00969                             1);
00970         atm->p[ip] = cache->iso_var[ip] * t;
00971     }
00972
00973     /* Restore potential temperature... */
00974     else if (ctl->isosurf == 3) {
00975         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
00976                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw,
00977                             1);
00978         atm->p[ip] = 1000. * pow(cache->iso_var[ip] / t, -1. / 0.286);
00979     }
00980
00981     /* Interpolate pressure... */
00982     else if (ctl->isosurf == 4) {
00983         if (atm->time[ip] <= cache->iso_ts[0])
00984             atm->p[ip] = cache->iso_ps[0];
00985         else if (atm->time[ip] >= cache->iso_ts[cache->iso_n - 1])
00986             atm->p[ip] = cache->iso_ps[cache->iso_n - 1];
00987         else {
00988             int idx = locate_irr(cache->iso_ts, cache->iso_n, atm->
time[ip]);
00989             atm->p[ip] = LIN(cache->iso_ts[idx], cache->iso_ps[idx],
00990                             cache->iso_ts[idx + 1], cache->iso_ps[idx + 1],
00991                             atm->time[ip]);
00992         }
00993     }
00994 }
00995 }
00996
00997 /*****
00998
00999 void module_meteo(
01000     ctl_t * ctl,
01001     met_t * met0,
01002     met_t * met1,
01003     atm_t * atm) {
01004
01005     /* Check quantity flags... */
01006     if (ctl->qnt_tsts >= 0)
01007         if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
01008             ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
01009
01010 #ifdef _OPENACC
01011 #pragma acc data present(ctl,met0,met1,atm)
01012 #pragma acc parallel loop independent gang vector
01013 #else
01014 #pragma omp parallel for default(shared)
01015 #endif
01016     for (int ip = 0; ip < atm->np; ip++) {
01017
01018         double ps, pt, pc, pv, t, u, v, w, h2o, o3, lwc, iwc, z, cw[3];
01019
01020         int ci[3];
01021
01022         /* Interpolate meteorological data... */
01023         intpol_met_time_3d(met0, met0->z, met1, met1->z, atm->
time[ip],
01024                             atm->p[ip], atm->lon[ip], atm->lat[ip], &z, ci, cw, 1);
01025         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01026                             atm->p[ip], atm->lon[ip], atm->lat[ip], &t, ci, cw, 0);
01027         intpol_met_time_3d(met0, met0->u, met1, met1->u, atm->
time[ip],
01028                             atm->p[ip], atm->lon[ip], atm->lat[ip], &u, ci, cw, 0);
01029         intpol_met_time_3d(met0, met0->v, met1, met1->v, atm->
time[ip],
01030                             atm->p[ip], atm->lon[ip], atm->lat[ip], &v, ci, cw, 0);
01031         intpol_met_time_3d(met0, met0->w, met1, met1->w, atm->
time[ip],
01032                             atm->p[ip], atm->lon[ip], atm->lat[ip], &w, ci, cw, 0);
01033         intpol_met_time_3d(met0, met0->pv, met1, met1->pv, atm->
time[ip],
01034                             atm->p[ip], atm->lon[ip], atm->lat[ip], &pv, ci, cw,
01035                             0);
01036         intpol_met_time_3d(met0, met0->h2o, met1, met1->h2o, atm->
time[ip],
01037                             atm->p[ip], atm->lon[ip], atm->lat[ip], &h2o, ci, cw,
01038                             0);
01039         intpol_met_time_3d(met0, met0->o3, met1, met1->o3, atm->

```

```

    time[ip],
01040         atm->p[ip], atm->lon[ip], atm->lat[ip], &o3, ci, cw,
01041         0);
01042     intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
    time[ip],
01043         atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
01044         0);
01045     intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
    time[ip],
01046         atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
01047         0);
01048     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
    time[ip],
01049         atm->lon[ip], atm->lat[ip], &ps, ci, cw, 0);
01050     intpol_met_time_2d(met0, met0->pt, met1, met1->pt, atm->
    time[ip],
01051         atm->lon[ip], atm->lat[ip], &pt, ci, cw, 0);
01052     intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
    time[ip],
01053         atm->lon[ip], atm->lat[ip], &pc, ci, cw, 0);
01054
01055     /* Set surface pressure... */
01056     if (ctl->qnt_ps >= 0)
01057         atm->q[ctl->qnt_ps][ip] = ps;
01058
01059     /* Set tropopause pressure... */
01060     if (ctl->qnt_pt >= 0)
01061         atm->q[ctl->qnt_pt][ip] = pt;
01062
01063     /* Set pressure... */
01064     if (ctl->qnt_p >= 0)
01065         atm->q[ctl->qnt_p][ip] = atm->p[ip];
01066
01067     /* Set geopotential height... */
01068     if (ctl->qnt_z >= 0)
01069         atm->q[ctl->qnt_z][ip] = z;
01070
01071     /* Set temperature... */
01072     if (ctl->qnt_t >= 0)
01073         atm->q[ctl->qnt_t][ip] = t;
01074
01075     /* Set zonal wind... */
01076     if (ctl->qnt_u >= 0)
01077         atm->q[ctl->qnt_u][ip] = u;
01078
01079     /* Set meridional wind... */
01080     if (ctl->qnt_v >= 0)
01081         atm->q[ctl->qnt_v][ip] = v;
01082
01083     /* Set vertical velocity... */
01084     if (ctl->qnt_w >= 0)
01085         atm->q[ctl->qnt_w][ip] = w;
01086
01087     /* Set water vapor vmr... */
01088     if (ctl->qnt_h2o >= 0)
01089         atm->q[ctl->qnt_h2o][ip] = h2o;
01090
01091     /* Set ozone vmr... */
01092     if (ctl->qnt_o3 >= 0)
01093         atm->q[ctl->qnt_o3][ip] = o3;
01094
01095     /* Set cloud liquid water content... */
01096     if (ctl->qnt_lwc >= 0)
01097         atm->q[ctl->qnt_lwc][ip] = lwc;
01098
01099     /* Set cloud ice water content... */
01100     if (ctl->qnt_iwc >= 0)
01101         atm->q[ctl->qnt_iwc][ip] = iwc;
01102
01103     /* Set cloud top pressure... */
01104     if (ctl->qnt_pc >= 0)
01105         atm->q[ctl->qnt_pc][ip] = pc;
01106
01107     /* Set nitric acid vmr... */
01108     if (ctl->qnt_hno3 >= 0)
01109         atm->q[ctl->qnt_hno3][ip] =
01110             clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip]);
01111
01112     /* Set hydroxyl number concentration... */
01113     if (ctl->qnt_oh >= 0)
01114         atm->q[ctl->qnt_oh][ip] =
01115             clim_oh(atm->time[ip], atm->lat[ip], atm->p[ip]);
01116
01117     /* Calculate horizontal wind... */
01118     if (ctl->qnt_vh >= 0)
01119         atm->q[ctl->qnt_vh][ip] = sqrt(u * u + v * v);
01120

```

```

01121      /* Calculate vertical velocity... */
01122      if (ctl->qnt_vz >= 0)
01123          atm->q[ctl->qnt_vz][ip] = -1e3 * H0 / atm->p[ip] * w;
01124
01125      /* Calculate relative humidity... */
01126      if (ctl->qnt_rh >= 0)
01127          atm->q[ctl->qnt_rh][ip] = RH(atm->p[ip], t, h2o);
01128
01129      /* Calculate potential temperature... */
01130      if (ctl->qnt_theta >= 0)
01131          atm->q[ctl->qnt_theta][ip] = THETA(atm->p[ip], t);
01132
01133      /* Set potential vorticity... */
01134      if (ctl->qnt_pv >= 0)
01135          atm->q[ctl->qnt_pv][ip] = pv;
01136
01137      /* Calculate T_ice (Marti and Mauersberger, 1993)... */
01138      if (ctl->qnt_tice >= 0)
01139          atm->q[ctl->qnt_tice][ip] =
01140              -2663.5 /
01141              (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
01142               12.537);
01143
01144      /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
01145      if (ctl->qnt_tnat >= 0) {
01146          double p_hno3;
01147          if (ctl->psc_hno3 > 0)
01148              p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
01149          else
01150              p_hno3 = clim_hno3(atm->time[ip], atm->lat[ip], atm->p[ip])
01151                  * 1e-9 * atm->p[ip] / 1.333224;
01152          double p_h2o =
01153              (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
01154          double a = 0.009179 - 0.00088 * log10(p_h2o);
01155          double b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
01156          double c = -11397.0 / a;
01157          double x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
01158          double x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
01159          if (x1 > 0)
01160              atm->q[ctl->qnt_tnat][ip] = x1;
01161          if (x2 > 0)
01162              atm->q[ctl->qnt_tnat][ip] = x2;
01163      }
01164
01165      /* Calculate T_STS (mean of T_ice and T_NAT)... */
01166      if (ctl->qnt_tsts >= 0)
01167          atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
01168                                           + atm->q[ctl->qnt_tnat][ip]);
01169      }
01170 }
01171
01172 /*****
01173
01174 void module_position(
01175     met_t * met0,
01176     met_t * met1,
01177     atm_t * atm,
01178     double *dt) {
01179
01180     #ifdef _OPENACC
01181     #pragma acc data present(met0,met1,atm,dt)
01182     #pragma acc parallel loop independent gang vector
01183     #else
01184     #pragma omp parallel for default(shared)
01185     #endif
01186     for (int ip = 0; ip < atm->np; ip++)
01187         if (dt[ip] != 0) {
01188
01189             double ps, cw[3];
01190
01191             int ci[3];
01192
01193             /* Calculate modulo... */
01194             atm->lon[ip] = FMOD(atm->lon[ip], 360.);
01195             atm->lat[ip] = FMOD(atm->lat[ip], 360.);
01196
01197             /* Check latitude... */
01198             while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01199                 if (atm->lat[ip] > 90) {
01200                     atm->lat[ip] = 180 - atm->lat[ip];
01201                     atm->lon[ip] += 180;
01202                 }
01203                 if (atm->lat[ip] < -90) {
01204                     atm->lat[ip] = -180 - atm->lat[ip];
01205                     atm->lon[ip] += 180;
01206                 }
01207             }

```



```

01208
01209 /* Check longitude... */
01210 while (atm->lon[ip] < -180)
01211     atm->lon[ip] += 360;
01212 while (atm->lon[ip] >= 180)
01213     atm->lon[ip] -= 360;
01214
01215 /* Check pressure... */
01216 if (atm->p[ip] < met0->p[met0->np - 1])
01217     atm->p[ip] = met0->p[met0->np - 1];
01218 else if (atm->p[ip] > 300.) {
01219     intpol_met_time_2d(met0, met0->ps, met1, met1->ps, atm->
time[ip],
01220                       atm->lon[ip], atm->lat[ip], &ps, ci, cw, 1);
01221     if (atm->p[ip] > ps)
01222         atm->p[ip] = ps;
01223 }
01224 }
01225 }
01226
01227 /*****
01228
01229 void module_sedi(
01230     ctl_t * ctl,
01231     met_t * met0,
01232     met_t * met1,
01233     atm_t * atm,
01234     double *dt) {
01235
01236 #ifdef _OPENACC
01237 #pragma acc data present(ctl,met0,met1,atm,dt)
01238 #pragma acc parallel loop independent gang vector
01239 #else
01240 #pragma omp parallel for default(shared)
01241 #endif
01242 for (int ip = 0; ip < atm->np; ip++)
01243     if (dt[ip] != 0) {
01244
01245         double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p, cw[3];
01246
01247         int ci[3];
01248
01249         /* Convert units... */
01250         p = 100. * atm->p[ip];
01251         r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01252         rho_p = atm->q[ctl->qnt_rho][ip];
01253
01254         /* Get temperature... */
01255         intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01256                           atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
1);
01257
01258         /* Density of dry air... */
01259         rho = p / (RA * T);
01260
01261         /* Dynamic viscosity of air... */
01262         eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01263
01264         /* Thermal velocity of an air molecule... */
01265         v = sqrt(8. * KB * T / (M_PI * 4.8096e-26));
01266
01267         /* Mean free path of an air molecule... */
01268         lambda = 2. * eta / (rho * v);
01269
01270         /* Knudsen number for air... */
01271         K = lambda / r_p;
01272
01273         /* Cunningham slip-flow correction... */
01274         G = 1. + K * (1.249 + 0.42 * exp(-0.87 / K));
01275
01276         /* Sedimentation (fall) velocity... */
01277         v_p = 2. * SQR(r_p) * (rho_p - rho) * G0 / (9. * eta) * G;
01278
01279         /* Calculate pressure change... */
01280         atm->p[ip] += DZ2DP(v_p * dt[ip] / 1000., atm->p[ip]);
01281     }
01282 }
01283 }
01284
01285 /*****
01286
01287 void module_oh_chem(
01288     ctl_t * ctl,
01289     met_t * met0,
01290     met_t * met1,
01291     atm_t * atm,
01292     double *dt) {

```

```

01293
01294  /* Check quantity flags... */
01295  if (ctl->qnt_m < 0)
01296      ERRMSG("Module needs quantity mass!");
01297
01298 #ifdef _OPENACC
01299 #pragma acc data present(ctl,atm,dt)
01300 #pragma acc parallel loop independent gang vector
01301 #else
01302 #pragma omp parallel for default(shared)
01303 #endif
01304  for (int ip = 0; ip < atm->np; ip++)
01305      if (dt[ip] != 0) {
01306
01307          double c, k, k0, ki, M, T, cw[3];
01308
01309          int ci[3];
01310
01311          /* Get temperature... */
01312          intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01313                          atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
1);
01314
01315          /* Calculate molecular density... */
01316          M = 7.243e21 * (atm->p[ip] / P0) / T;
01317
01318          /* Calculate rate coefficient for X + OH + M -> XOH + M
(JPL Publication 15-10) ... */
01319          k0 = ctl->oh_chem[0] *
01320              (ctl->oh_chem[1] > 0 ? pow(T / 300., -ctl->oh_chem[1]) : 1.);
01321          ki = ctl->oh_chem[2] *
01322              (ctl->oh_chem[3] > 0 ? pow(T / 300., -ctl->oh_chem[3]) : 1.);
01323          c = log10(k0 * M / ki);
01324          k = k0 * M / (1. + k0 * M / ki) * pow(0.6, 1. / (1. + c * c));
01325
01326          /* Calculate exponential decay... */
01327          atm->q[ctl->qnt_m][ip] *=
01328              exp(-dt[ip] * k * clim_oh(atm->time[ip], atm->lat[ip], atm->
p[ip]));
01329      }
01330  }
01331 }
01332
01333
01334 /*****
01335
01336 void module_wet_deposition(
01337     ctl_t * ctl,
01338     met_t * met0,
01339     met_t * met1,
01340     atm_t * atm,
01341     double *dt) {
01342
01343     /* Check quantity flags... */
01344     if (ctl->qnt_m < 0)
01345         ERRMSG("Module needs quantity mass!");
01346
01347 #ifdef _OPENACC
01348 #pragma acc data present(ctl,atm,dt)
01349 #pragma acc parallel loop independent gang vector
01350 #else
01351 #pragma omp parallel for default(shared)
01352 #endif
01353     for (int ip = 0; ip < atm->np; ip++)
01354         if (dt[ip] != 0) {
01355
01356             double H, Is, Si, T, cl, lambda, iwc, lwc, pc, cw[3];
01357
01358             int inside, ci[3];
01359
01360             /* Check whether particle is below cloud top... */
01361             intpol_met_time_2d(met0, met0->pc, met1, met1->pc, atm->
time[ip],
01362                             atm->lon[ip], atm->lat[ip], &pc, ci, cw, 1);
01363             if (!isfinite(pc) || atm->p[ip] <= pc)
01364                 continue;
01365
01366             /* Check whether particle is inside or below cloud... */
01367             intpol_met_time_3d(met0, met0->lwc, met1, met1->lwc, atm->
time[ip],
01368                             atm->p[ip], atm->lon[ip], atm->lat[ip], &lwc, ci, cw,
1);
01369             intpol_met_time_3d(met0, met0->iwc, met1, met1->iwc, atm->
time[ip],
01370                             atm->p[ip], atm->lon[ip], atm->lat[ip], &iwc, ci, cw,
0);
01371             inside = (iwc > 0 || lwc > 0);
01372
01373
01374

```

```

01375      /* Estimate precipitation rate (Pisso et al., 2019)... */
01376      intpol_met_time_2d(met0, met0->cl, met1, met1->cl, atm->
time[ip],
01377                      atm->lon[ip], atm->lat[ip], &cl, ci, cw, 0);
01378      Is = pow(2. * cl, 1. / 0.36);
01379      if (Is < 0.01)
01380          continue;
01381
01382      /* Calculate in-cloud scavenging for gases... */
01383      if (inside) {
01384
01385          /* Get temperature... */
01386          intpol_met_time_3d(met0, met0->t, met1, met1->t, atm->
time[ip],
01387                          atm->p[ip], atm->lon[ip], atm->lat[ip], &T, ci, cw,
0);
01388
01389          /* Get Henry's constant (Sander, 2015)... */
01390          H = ctl->wet_depo[2] * 101.325
01391              * exp(ctl->wet_depo[3] * (1. / T - 1. / 298.15));
01392
01393          /* Get scavenging coefficient (Hertel et al., 1995)... */
01394          Si = 1. / ((1. - cl) / (H * RI / P0 * T) + cl);
01395          lambda = 6.2 * Si * Is / 3.6e6;
01396      }
01397
01398      /* Calculate below-cloud scavenging for gases (Pisso et al., 2019)... */
01399      else
01400          lambda = ctl->wet_depo[0] * pow(Is, ctl->wet_depo[1]);
01401
01402      /* Calculate exponential decay... */
01403      atm->q[ctl->qnt_m][ip] *= exp(-dt[ip] * lambda);
01404  }
01405 }
01406 }
01407
01408 /*****
01409
01410 void write_output (
01411     const char *dirname,
01412     ctl_t * ctl,
01413     met_t * met0,
01414     met_t * met1,
01415     atm_t * atm,
01416     double t) {
01417
01418     char filename[2 * LEN];
01419
01420     double r;
01421
01422     int year, mon, day, hour, min, sec, updated = 0;
01423
01424     /* Get time... */
01425     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01426
01427     /* Write atmospheric data... */
01428     if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01429         if (!updated) {
01430             #ifdef USE_NVTX
01431                 RANGE_PUSH("W atm D2H", RED);
01432             #endif
01433             #ifdef _OPENACC
01434                 #pragma acc update host (atm[:1])
01435             #endif
01436             #ifdef USE_NVTX
01437                 RANGE_POP;
01438             #endif
01439             updated = 1;
01440         }
01441         #ifdef USE_NVTX
01442             RANGE_PUSH("IO", YELLOW);
01443         #endif
01444         sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01445                 dirname, ctl->atm_basename, year, mon, day, hour, min);
01446         write_atm(filename, ctl, atm, t);
01447         #ifdef USE_NVTX
01448             RANGE_POP;
01449         #endif
01450     }
01451
01452     /* Write gridded data... */
01453     if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01454         if (!updated) {
01455             #ifdef USE_NVTX
01456                 RANGE_PUSH("W grd D2H", RED);
01457             #endif
01458             #ifdef _OPENACC
01459                 #pragma acc update host (atm[:1])

```

```

01460 #endif
01461 #ifdef USE_NVTX
01462     RANGE_POP;
01463 #endif
01464     updated = 1;
01465 }
01466 #ifdef USE_NVTX
01467     RANGE_PUSH("IO", YELLOW);
01468 #endif
01469     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d.tab",
01470             dirname, ctl->grid_basename, year, mon, day, hour, min);
01471     write_grid(filename, ctl, met0, met1, atm, t);
01472 #ifdef USE_NVTX
01473     RANGE_POP;
01474 #endif
01475 }
01476
01477 /* Write CSI data... */
01478 if (ctl->csi_basename[0] != '-') {
01479     if (!updated) {
01480 #ifdef USE_NVTX
01481         RANGE_PUSH("W csi D2H", RED);
01482 #endif
01483 #ifdef _OPENACC
01484 #pragma acc update host(atm[:1])
01485 #endif
01486 #ifdef USE_NVTX
01487         RANGE_POP;
01488 #endif
01489         updated = 1;
01490     }
01491 #ifdef USE_NVTX
01492     RANGE_PUSH("IO", YELLOW);
01493 #endif
01494     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01495     write_csi(filename, ctl, atm, t);
01496 #ifdef USE_NVTX
01497     RANGE_POP;
01498 #endif
01499 }
01500
01501 /* Write ensemble data... */
01502 if (ctl->ens_basename[0] != '-') {
01503     if (!updated) {
01504 #ifdef USE_NVTX
01505         RANGE_PUSH("W csi D2H", RED);
01506 #endif
01507 #ifdef _OPENACC
01508 #pragma acc update host(atm[:1])
01509 #endif
01510 #ifdef USE_NVTX
01511         RANGE_POP;
01512 #endif
01513         updated = 1;
01514     }
01515 #ifdef USE_NVTX
01516     RANGE_PUSH("IO", YELLOW);
01517 #endif
01518     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01519     write_ens(filename, ctl, atm, t);
01520 #ifdef USE_NVTX
01521     RANGE_POP;
01522 #endif
01523 }
01524
01525 /* Write profile data... */
01526 if (ctl->prof_basename[0] != '-') {
01527     if (!updated) {
01528 #ifdef USE_NVTX
01529         RANGE_PUSH("W prof D2H", RED);
01530 #endif
01531 #ifdef _OPENACC
01532 #pragma acc update host(atm[:1])
01533 #endif
01534 #ifdef USE_NVTX
01535         RANGE_POP;
01536 #endif
01537         updated = 1;
01538     }
01539 #ifdef USE_NVTX
01540     RANGE_PUSH("IO", YELLOW);
01541 #endif
01542     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01543     write_prof(filename, ctl, met0, met1, atm, t);
01544 #ifdef USE_NVTX
01545     RANGE_POP;
01546 #endif

```

```

01547     }
01548
01549     /* Write station data... */
01550     if (ctl->stat_basename[0] != '-') {
01551         if (!updated) {
01552             #ifdef USE_NVTX
01553                 RANGE_PUSH("W st D2H", RED);
01554             #endif
01555             #ifdef _OPENACC
01556             #pragma acc update host (atm[:1])
01557             #endif
01558             #ifdef USE_NVTX
01559                 RANGE_POP;
01560             #endif
01561             updated = 1;
01562         }
01563         #ifdef USE_NVTX
01564             RANGE_PUSH("IO", YELLOW);
01565         #endif
01566         sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01567         write_station(filename, ctl, atm, t);
01568         #ifdef USE_NVTX
01569             RANGE_POP;
01570         #endif
01571     }
01572 }

```

5.37 tropo.c File Reference

Create tropopause climatology from meteorological data.

Functions

- void [add_text_attribute](#) (int ncid, char *varname, char *attrname, char *text)
- int [main](#) (int argc, char *argv[])

5.37.1 Detailed Description

Create tropopause climatology from meteorological data.

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

5.37.2 Function Documentation

5.37.2.1 void add_text_attribute (int ncid, char * varname, char * attrname, char * text)

Definition at line [337](#) of file [tropo.c](#).

```

00341     {
00342
00343     int varid;
00344
00345     NC(nc_inq_varid(ncid, varname, &varid));
00346     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00347 }

```

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

Definition at line 41 of file tropo.c.

```

00043         {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00050         lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];
00051
00052     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00053         clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00054         wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];
00055
00056     static size_t count[10], start[10];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00068     lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00069     dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00073     h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00074
00075     /* Loop over files... */
00076     for (i = 3; i < argc; i++) {
00077
00078         /* Read meteorological data... */
00079         ctl.met_tropo = 0;
00080         if (!read_met(&ctl, argv[i], met))
00081             continue;
00082
00083         /* Set horizontal grid... */
00084         if (!init) {
00085             init = 1;
00086
00087             /* Get grid... */
00088             if (dlon <= 0)
00089                 dlon = fabs(met->lon[1] - met->lon[0]);
00090             if (dlat <= 0)
00091                 dlat = fabs(met->lat[1] - met->lat[0]);
00092             if (lon0 < -360 && lon1 > 360) {
00093                 lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00094                 lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00095             }
00096             nx = ny = 0;
00097             for (lon = lon0; lon <= lon1; lon += dlon) {
00098                 lons[nx] = lon;
00099                 if ((++nx) > EX)
00100                     ERRMSG("Too many longitudes!");
00101             }
00102             if (lat0 < -90 && lat1 > 90) {
00103                 lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00104                 lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00105             }
00106             for (lat = lat0; lat <= lat1; lat += dlat) {
00107                 lats[ny] = lat;
00108                 if ((++ny) > EY)
00109                     ERRMSG("Too many latitudes!");
00110             }
00111
00112             /* Create netCDF file... */
00113             printf("Write tropopause data file: %s\n", argv[2]);
00114             NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00115
00116             /* Create dimensions... */
00117             NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00118             NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00119             NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00120
00121             /* Create variables... */
00122             NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));

```

```

00123     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00124     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00125     NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00126     NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00127     NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00128     if (h2o)
00129         NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00130     NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00131     NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dypid));
00132     NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00133     if (h2o)
00134         NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00135     NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00136     NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00137     NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00138     if (h2o)
00139         NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00140     NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00141     NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00142     NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00143     if (h2o)
00144         NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00145
00146     /* Set attributes... */
00147     add_text_attribute(ncid, "time", "units",
00148                       "seconds since 2000-01-01 00:00:00 UTC");
00149     add_text_attribute(ncid, "time", "long_name", "time");
00150     add_text_attribute(ncid, "lon", "units", "degrees_east");
00151     add_text_attribute(ncid, "lon", "long_name", "longitude");
00152     add_text_attribute(ncid, "lat", "units", "degrees_north");
00153     add_text_attribute(ncid, "lat", "long_name", "latitude");
00154
00155     add_text_attribute(ncid, "clp_z", "units", "km");
00156     add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00157     add_text_attribute(ncid, "clp_p", "units", "hPa");
00158     add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00159     add_text_attribute(ncid, "clp_t", "units", "K");
00160     add_text_attribute(ncid, "clp_t", "long_name",
00161                       "cold point temperature");
00162     if (h2o) {
00163         add_text_attribute(ncid, "clp_q", "units", "ppv");
00164         add_text_attribute(ncid, "clp_q", "long_name",
00165                           "cold point water vapor");
00166     }
00167
00168     add_text_attribute(ncid, "dyn_z", "units", "km");
00169     add_text_attribute(ncid, "dyn_z", "long_name",
00170                       "dynamical tropopause height");
00171     add_text_attribute(ncid, "dyn_p", "units", "hPa");
00172     add_text_attribute(ncid, "dyn_p", "long_name",
00173                       "dynamical tropopause pressure");
00174     add_text_attribute(ncid, "dyn_t", "units", "K");
00175     add_text_attribute(ncid, "dyn_t", "long_name",
00176                       "dynamical tropopause temperature");
00177     if (h2o) {
00178         add_text_attribute(ncid, "dyn_q", "units", "ppv");
00179         add_text_attribute(ncid, "dyn_q", "long_name",
00180                           "dynamical tropopause water vapor");
00181     }
00182
00183     add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00184     add_text_attribute(ncid, "wmo_1st_z", "long_name",
00185                       "WMO 1st tropopause height");
00186     add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00187     add_text_attribute(ncid, "wmo_1st_p", "long_name",
00188                       "WMO 1st tropopause pressure");
00189     add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00190     add_text_attribute(ncid, "wmo_1st_t", "long_name",
00191                       "WMO 1st tropopause temperature");
00192     if (h2o) {
00193         add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
00194         add_text_attribute(ncid, "wmo_1st_q", "long_name",
00195                           "WMO 1st tropopause water vapor");
00196     }
00197
00198     add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00199     add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00200                       "WMO 2nd tropopause height");
00201     add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00202     add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00203                       "WMO 2nd tropopause pressure");
00204     add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00205     add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00206                       "WMO 2nd tropopause temperature");
00207     if (h2o) {
00208         add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00209         add_text_attribute(ncid, "wmo_2nd_q", "long_name",

```

```

00210             "WMO 2nd tropopause water vapor");
00211     }
00212
00213     /* End definition... */
00214     NC(nc_enddef(ncid));
00215
00216     /* Write longitude and latitude... */
00217     NC(nc_put_var_double(ncid, latid, lats));
00218     NC(nc_put_var_double(ncid, lonid, lons));
00219 }
00220
00221 /* Write time... */
00222 start[0] = (size_t) nt;
00223 count[0] = 1;
00224 start[1] = 0;
00225 count[1] = (size_t) ny;
00226 start[2] = 0;
00227 count[2] = (size_t) nx;
00228 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00229
00230 /* Get cold point... */
00231 ctl.met_tropo = 2;
00232 read_met_tropo(&ctl, met);
00233 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00234 for (ix = 0; ix < nx; ix++)
00235     for (iy = 0; iy < ny; iy++) {
00236         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00237                             &pt[iy * nx + ix], ci, cw, 1);
00238         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00239                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00240         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00241                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00242         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00243                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00244     }
00245
00246 /* Write data... */
00247 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00248 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00249 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00250 if (h2o)
00251     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00252
00253 /* Get dynamical tropopause... */
00254 ctl.met_tropo = 5;
00255 read_met_tropo(&ctl, met);
00256 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00257 for (ix = 0; ix < nx; ix++)
00258     for (iy = 0; iy < ny; iy++) {
00259         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00260                             &pt[iy * nx + ix], ci, cw, 1);
00261         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00262                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00263         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00264                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00265         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00266                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00267     }
00268
00269 /* Write data... */
00270 NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00271 NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00272 NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00273 if (h2o)
00274     NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00275
00276 /* Get WMO 1st tropopause... */
00277 ctl.met_tropo = 3;
00278 read_met_tropo(&ctl, met);
00279 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00280 for (ix = 0; ix < nx; ix++)
00281     for (iy = 0; iy < ny; iy++) {
00282         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00283                             &pt[iy * nx + ix], ci, cw, 1);
00284         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00285                             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00286         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00287                             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00288         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00289                             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00290     }
00291
00292 /* Write data... */
00293 NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00294 NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00295 NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00296 if (h2o)

```

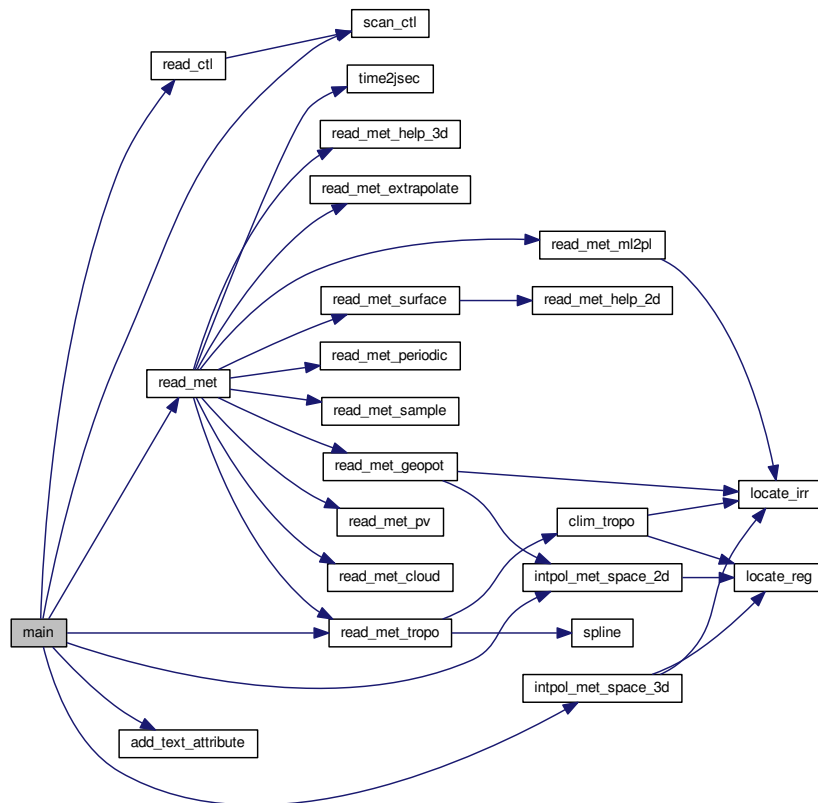


```

00297     NC(nc_put_vara_double(ncid, wmo1qid, start, count, qt));
00298
00299     /* Get WMO 2nd tropopause... */
00300     ctl.met_tropo = 4;
00301     read_met_tropo(&ctl, met);
00302 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00303     for (ix = 0; ix < nx; ix++)
00304         for (iy = 0; iy < ny; iy++) {
00305             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00306                                &pt[iy * nx + ix], ci, cw, 1);
00307             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00308                                lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00309             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00310                                lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00311             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00312                                lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00313         }
00314
00315     /* Write data... */
00316     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00317     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00318     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00319     if (h2o)
00320         NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00321
00322     /* Increment time step counter... */
00323     nt++;
00324 }
00325
00326 /* Close file... */
00327 NC(nc_close(ncid));
00328
00329 /* Free... */
00330 free(met);
00331
00332 return EXIT_SUCCESS;
00333 }

```

Here is the call graph for this function:



5.38 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) 2013-2019 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* -----
00028   Functions...
00029   ----- */
00030
00031 void add_text_attribute(
00032     int ncid,
00033     char *varname,
00034     char *attrname,
00035     char *text);
00036
00037 /* -----
00038   Main...
00039   ----- */
00040
00041 int main(
00042     int argc,
00043     char *argv[]) {
00044
00045     ctl_t ctl;
00046
00047     met_t *met;
00048
00049     static double pt[EX * EY], qt[EX * EY], zt[EX * EY], tt[EX * EY], lon, lon0,
00050         lon1, lons[EX], dlon, lat, lat0, lat1, lats[EY], dlat, cw[3];
00051
00052     static int init, i, ix, iy, nx, ny, nt, ncid, dims[3], timid, lonid, latid,
00053         clppid, clpqid, clptid, clpzid, dynpid, dynqid, dyntid, dynzid, wmolpid,
00054         wmolqid, wmoltid, wmolzid, wmo2pid, wmo2qid, wmo2tid, wmo2zid, h2o, ci[3];
00055
00056     static size_t count[10], start[10];
00057
00058     /* Allocate... */
00059     ALLOC(met, met_t, 1);
00060
00061     /* Check arguments... */
00062     if (argc < 4)
00063         ERRMSG("Give parameters: <ctl> <tropo.nc> <met0> [ <met1> ... ]");
00064
00065     /* Read control parameters... */
00066     read_ctl(argv[1], argc, argv, &ctl);
00067     lon0 = scan_ctl(argv[1], argc, argv, "TROPO_LON0", -1, "-180", NULL);
00068     lon1 = scan_ctl(argv[1], argc, argv, "TROPO_LON1", -1, "180", NULL);
00069     dlon = scan_ctl(argv[1], argc, argv, "TROPO_DLON", -1, "-999", NULL);
00070     lat0 = scan_ctl(argv[1], argc, argv, "TROPO_LAT0", -1, "-90", NULL);
00071     lat1 = scan_ctl(argv[1], argc, argv, "TROPO_LAT1", -1, "90", NULL);
00072     dlat = scan_ctl(argv[1], argc, argv, "TROPO_DLAT", -1, "-999", NULL);
00073     h2o = (int) scan_ctl(argv[1], argc, argv, "TROPO_H2O", -1, "1", NULL);
00074
00075     /* Loop over files... */
00076     for (i = 3; i < argc; i++) {
00077
00078         /* Read meteorological data... */
00079         ctl.met_tropo = 0;
00080         if (!read_met(&ctl, argv[i], met))
00081             continue;
00082
00083         /* Set horizontal grid... */
00084         if (!init) {
00085             init = 1;
00086
00087             /* Get grid... */
00088             if (dlon <= 0)
00089                 dlon = fabs(met->lon[1] - met->lon[0]);

```

```

00090     if (dlat <= 0)
00091         dlat = fabs(met->lat[1] - met->lat[0]);
00092     if (lon0 < -360 && lon1 > 360) {
00093         lon0 = gsl_stats_min(met->lon, 1, (size_t) met->nx);
00094         lon1 = gsl_stats_max(met->lon, 1, (size_t) met->nx);
00095     }
00096     nx = ny = 0;
00097     for (lon = lon0; lon <= lon1; lon += dlon) {
00098         lons[nx] = lon;
00099         if ((++nx) > EX)
00100             ERRMSG("Too many longitudes!");
00101     }
00102     if (lat0 < -90 && lat1 > 90) {
00103         lat0 = gsl_stats_min(met->lat, 1, (size_t) met->ny);
00104         lat1 = gsl_stats_max(met->lat, 1, (size_t) met->ny);
00105     }
00106     for (lat = lat0; lat <= lat1; lat += dlat) {
00107         lats[ny] = lat;
00108         if ((++ny) > EY)
00109             ERRMSG("Too many latitudes!");
00110     }
00111
00112     /* Create netCDF file... */
00113     printf("Write tropopause data file: %s\n", argv[2]);
00114     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00115
00116     /* Create dimensions... */
00117     NC(nc_def_dim(ncid, "time", (size_t) NC_UNLIMITED, &dims[0]));
00118     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[1]));
00119     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[2]));
00120
00121     /* Create variables... */
00122     NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00123     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[1], &latid));
00124     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[2], &lonid));
00125     NC(nc_def_var(ncid, "clp_z", NC_FLOAT, 3, &dims[0], &clpzid));
00126     NC(nc_def_var(ncid, "clp_p", NC_FLOAT, 3, &dims[0], &clppid));
00127     NC(nc_def_var(ncid, "clp_t", NC_FLOAT, 3, &dims[0], &clptid));
00128     if (h2o)
00129         NC(nc_def_var(ncid, "clp_q", NC_FLOAT, 3, &dims[0], &clpqid));
00130     NC(nc_def_var(ncid, "dyn_z", NC_FLOAT, 3, &dims[0], &dynzid));
00131     NC(nc_def_var(ncid, "dyn_p", NC_FLOAT, 3, &dims[0], &dynpid));
00132     NC(nc_def_var(ncid, "dyn_t", NC_FLOAT, 3, &dims[0], &dyntid));
00133     if (h2o)
00134         NC(nc_def_var(ncid, "dyn_q", NC_FLOAT, 3, &dims[0], &dynqid));
00135     NC(nc_def_var(ncid, "wmo_1st_z", NC_FLOAT, 3, &dims[0], &wmo1zid));
00136     NC(nc_def_var(ncid, "wmo_1st_p", NC_FLOAT, 3, &dims[0], &wmo1pid));
00137     NC(nc_def_var(ncid, "wmo_1st_t", NC_FLOAT, 3, &dims[0], &wmo1tid));
00138     if (h2o)
00139         NC(nc_def_var(ncid, "wmo_1st_q", NC_FLOAT, 3, &dims[0], &wmo1qid));
00140     NC(nc_def_var(ncid, "wmo_2nd_z", NC_FLOAT, 3, &dims[0], &wmo2zid));
00141     NC(nc_def_var(ncid, "wmo_2nd_p", NC_FLOAT, 3, &dims[0], &wmo2pid));
00142     NC(nc_def_var(ncid, "wmo_2nd_t", NC_FLOAT, 3, &dims[0], &wmo2tid));
00143     if (h2o)
00144         NC(nc_def_var(ncid, "wmo_2nd_q", NC_FLOAT, 3, &dims[0], &wmo2qid));
00145
00146     /* Set attributes... */
00147     add_text_attribute(ncid, "time", "units",
00148         "seconds since 2000-01-01 00:00:00 UTC");
00149     add_text_attribute(ncid, "time", "long_name", "time");
00150     add_text_attribute(ncid, "lon", "units", "degrees_east");
00151     add_text_attribute(ncid, "lon", "long_name", "longitude");
00152     add_text_attribute(ncid, "lat", "units", "degrees_north");
00153     add_text_attribute(ncid, "lat", "long_name", "latitude");
00154
00155     add_text_attribute(ncid, "clp_z", "units", "km");
00156     add_text_attribute(ncid, "clp_z", "long_name", "cold point height");
00157     add_text_attribute(ncid, "clp_p", "units", "hPa");
00158     add_text_attribute(ncid, "clp_p", "long_name", "cold point pressure");
00159     add_text_attribute(ncid, "clp_t", "units", "K");
00160     add_text_attribute(ncid, "clp_t", "long_name",
00161         "cold point temperature");
00162     if (h2o) {
00163         add_text_attribute(ncid, "clp_q", "units", "ppv");
00164         add_text_attribute(ncid, "clp_q", "long_name",
00165             "cold point water vapor");
00166     }
00167
00168     add_text_attribute(ncid, "dyn_z", "units", "km");
00169     add_text_attribute(ncid, "dyn_z", "long_name",
00170         "dynamical tropopause height");
00171     add_text_attribute(ncid, "dyn_p", "units", "hPa");
00172     add_text_attribute(ncid, "dyn_p", "long_name",
00173         "dynamical tropopause pressure");
00174     add_text_attribute(ncid, "dyn_t", "units", "K");
00175     add_text_attribute(ncid, "dyn_t", "long_name",
00176         "dynamical tropopause temperature");

```

```

00177     if (h2o) {
00178         add_text_attribute(ncid, "dyn_q", "units", "ppv");
00179         add_text_attribute(ncid, "dyn_q", "long_name",
00180             "dynamical tropopause water vapor");
00181     }
00182
00183     add_text_attribute(ncid, "wmo_1st_z", "units", "km");
00184     add_text_attribute(ncid, "wmo_1st_z", "long_name",
00185         "WMO 1st tropopause height");
00186     add_text_attribute(ncid, "wmo_1st_p", "units", "hPa");
00187     add_text_attribute(ncid, "wmo_1st_p", "long_name",
00188         "WMO 1st tropopause pressure");
00189     add_text_attribute(ncid, "wmo_1st_t", "units", "K");
00190     add_text_attribute(ncid, "wmo_1st_t", "long_name",
00191         "WMO 1st tropopause temperature");
00192     if (h2o) {
00193         add_text_attribute(ncid, "wmo_1st_q", "units", "ppv");
00194         add_text_attribute(ncid, "wmo_1st_q", "long_name",
00195             "WMO 1st tropopause water vapor");
00196     }
00197
00198     add_text_attribute(ncid, "wmo_2nd_z", "units", "km");
00199     add_text_attribute(ncid, "wmo_2nd_z", "long_name",
00200         "WMO 2nd tropopause height");
00201     add_text_attribute(ncid, "wmo_2nd_p", "units", "hPa");
00202     add_text_attribute(ncid, "wmo_2nd_p", "long_name",
00203         "WMO 2nd tropopause pressure");
00204     add_text_attribute(ncid, "wmo_2nd_t", "units", "K");
00205     add_text_attribute(ncid, "wmo_2nd_t", "long_name",
00206         "WMO 2nd tropopause temperature");
00207     if (h2o) {
00208         add_text_attribute(ncid, "wmo_2nd_q", "units", "ppv");
00209         add_text_attribute(ncid, "wmo_2nd_q", "long_name",
00210             "WMO 2nd tropopause water vapor");
00211     }
00212
00213     /* End definition... */
00214     NC(nc_enddef(ncid));
00215
00216     /* Write longitude and latitude... */
00217     NC(nc_put_var_double(ncid, latid, lats));
00218     NC(nc_put_var_double(ncid, lonid, lons));
00219 }
00220
00221 /* Write time... */
00222 start[0] = (size_t) nt;
00223 count[0] = 1;
00224 start[1] = 0;
00225 count[1] = (size_t) ny;
00226 start[2] = 0;
00227 count[2] = (size_t) nx;
00228 NC(nc_put_vara_double(ncid, timid, start, count, &met->time));
00229
00230 /* Get cold point... */
00231 ctl.met_tropo = 2;
00232 read_met_tropo(&ctl, met);
00233 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00234 for (ix = 0; ix < nx; ix++)
00235     for (iy = 0; iy < ny; iy++) {
00236         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00237             &pt[iy * nx + ix], ci, cw, 1);
00238         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00239             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00240         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00241             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00242         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00243             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00244     }
00245
00246 /* Write data... */
00247 NC(nc_put_vara_double(ncid, clpzid, start, count, zt));
00248 NC(nc_put_vara_double(ncid, clppid, start, count, pt));
00249 NC(nc_put_vara_double(ncid, clptid, start, count, tt));
00250 if (h2o)
00251     NC(nc_put_vara_double(ncid, clpqid, start, count, qt));
00252
00253 /* Get dynamical tropopause... */
00254 ctl.met_tropo = 5;
00255 read_met_tropo(&ctl, met);
00256 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00257 for (ix = 0; ix < nx; ix++)
00258     for (iy = 0; iy < ny; iy++) {
00259         intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00260             &pt[iy * nx + ix], ci, cw, 1);
00261         intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00262             lats[iy], &zt[iy * nx + ix], ci, cw, 1);
00263         intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],

```

```

00264         lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00265         intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00266         lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00267     }
00268
00269     /* Write data... */
00270     NC(nc_put_vara_double(ncid, dynzid, start, count, zt));
00271     NC(nc_put_vara_double(ncid, dynpid, start, count, pt));
00272     NC(nc_put_vara_double(ncid, dyntid, start, count, tt));
00273     if (h2o)
00274         NC(nc_put_vara_double(ncid, dynqid, start, count, qt));
00275
00276     /* Get WMO 1st tropopause... */
00277     ctl.met_tropo = 3;
00278     read_met_tropo(&ctl, met);
00279 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00280     for (ix = 0; ix < nx; ix++)
00281         for (iy = 0; iy < ny; iy++) {
00282             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00283             &pt[iy * nx + ix], ci, cw, 1);
00284             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00285             lats[iy], &z[iy * nx + ix], ci, cw, 1);
00286             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00287             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00288             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00289             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00290         }
00291
00292     /* Write data... */
00293     NC(nc_put_vara_double(ncid, wmolzid, start, count, zt));
00294     NC(nc_put_vara_double(ncid, wmolpid, start, count, pt));
00295     NC(nc_put_vara_double(ncid, wmoltid, start, count, tt));
00296     if (h2o)
00297         NC(nc_put_vara_double(ncid, wmolqid, start, count, qt));
00298
00299     /* Get WMO 2nd tropopause... */
00300     ctl.met_tropo = 4;
00301     read_met_tropo(&ctl, met);
00302 #pragma omp parallel for default(shared) private(ix,iy,ci,cw)
00303     for (ix = 0; ix < nx; ix++)
00304         for (iy = 0; iy < ny; iy++) {
00305             intpol_met_space_2d(met, met->pt, lons[ix], lats[iy],
00306             &pt[iy * nx + ix], ci, cw, 1);
00307             intpol_met_space_3d(met, met->z, pt[iy * nx + ix], lons[ix],
00308             lats[iy], &z[iy * nx + ix], ci, cw, 1);
00309             intpol_met_space_3d(met, met->t, pt[iy * nx + ix], lons[ix],
00310             lats[iy], &tt[iy * nx + ix], ci, cw, 0);
00311             intpol_met_space_3d(met, met->h2o, pt[iy * nx + ix], lons[ix],
00312             lats[iy], &qt[iy * nx + ix], ci, cw, 0);
00313         }
00314
00315     /* Write data... */
00316     NC(nc_put_vara_double(ncid, wmo2zid, start, count, zt));
00317     NC(nc_put_vara_double(ncid, wmo2pid, start, count, pt));
00318     NC(nc_put_vara_double(ncid, wmo2tid, start, count, tt));
00319     if (h2o)
00320         NC(nc_put_vara_double(ncid, wmo2qid, start, count, qt));
00321
00322     /* Increment time step counter... */
00323     nt++;
00324 }
00325
00326 /* Close file... */
00327 NC(nc_close(ncid));
00328
00329 /* Free... */
00330 free(met);
00331
00332 return EXIT_SUCCESS;
00333 }
00334
00335 /*****
00336
00337 void add_text_attribute(
00338     int ncid,
00339     char *varname,
00340     char *attrname,
00341     char *text) {
00342
00343     int varid;
00344
00345     NC(nc_inq_varid(ncid, varname, &varid));
00346     NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00347 }

```

5.39 tropo_sample.c File Reference

Sample tropopause climatology.

Functions

- double [intpol_help](#) (double x0, double y0, double x1, double y1, double x)
- double [intpol_2d](#) (float array[EX][EY], double lons[EX], double lats[EY], size_t nlon, size_t nlat, double lon, double lat)
- int [main](#) (int argc, char *argv[])

5.39.1 Detailed Description

Sample tropopause climatology.

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

5.39.2 Function Documentation

5.39.2.1 double intpol_help (double x0, double y0, double x1, double y1, double x)

Definition at line 269 of file [tropo_sample.c](#).

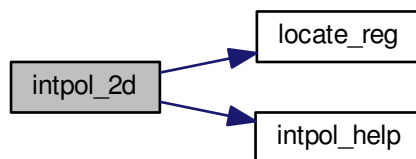
```
00274         {
00275
00276     /* Linear interpolation... */
00277     if (gsl_finite(y0) && gsl_finite(y1))
00278         return LIN(x0, y0, x1, y1, x);
00279
00280     /* Nearest neighbour... */
00281     else {
00282         if (fabs(x - x0) < fabs(x - x1))
00283             return y0;
00284         else
00285             return y1;
00286     }
00287 }
```

5.39.2.2 double intpol_2d (float array[EX][EY], double lons[EX], double lats[EY], size_t nlon, size_t nlat, double lon, double lat)

Definition at line 291 of file [tropo_sample.c](#).

```
00298         {
00299
00300     double aux0, aux1;
00301
00302     /* Adjust longitude... */
00303     if (lon < lons[0])
00304         lon += 360;
00305     else if (lon > lons[nlon - 1])
00306         lon -= 360;
00307
00308     /* Get indices... */
00309     int ix = locate_reg(lons, (int) nlon, lon);
00310     int iy = locate_reg(lats, (int) nlat, lat);
00311
00312     /* Interpolate in longitude... */
00313     aux0 = intpol_help(lons[ix], array[ix][iy],
00314                       lons[ix + 1], array[ix + 1][iy], lon);
00315     aux1 = intpol_help(lons[ix], array[ix][iy + 1],
00316                       lons[ix + 1], array[ix + 1][iy + 1], lon);
00317
00318     /* Interpolate in latitude... */
00319     return intpol_help(lats[iy], aux0, lats[iy + 1], aux1, lat);
00320 }
```

Here is the call graph for this function:



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

Definition at line 60 of file [tropo_sample.c](#).

```

00062         {
00063
00064     ctl_t ctl;
00065
00066     atm_t *atm;
00067
00068     static FILE *out;
00069
00070     static char varname[LEN];
00071
00072     static double times[NT], lons[EX], lats[EY], time0, time1, z0, z1, p0, p1,
00073         t0, t1, q0, q1;
00074
00075     static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00076         tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00077         tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00078
00079     static int ip, iq, it, it_old = -999, dimid[10], ncid,
00080         varid, varid_z, varid_p, varid_t, varid_q, h2o;
00081
00082     static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
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
00094     /* Read atmospheric data... */
00095     if (!read_atm(argv[5], &ctl, atm))
00096         ERRMSG("Cannot open file!");
00097
00098     /* Open tropopause file... */
00099     printf("Read tropopause data: %s\n", argv[3]);
00100     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00101         ERRMSG("Cannot open file!");
00102
00103     /* Get dimensions... */
00104     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00105     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00106     if (ntime > NT)
00107         ERRMSG("Too many times!");
00108     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00109     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00110     if (nlat > EY)
00111         ERRMSG("Too many latitudes!");
00112     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00113     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00114     if (nlon > EX)
00115         ERRMSG("Too many longitudes!");
00116

```

```

00117  /* Read coordinates... */
00118  NC(nc_inq_varid(ncid, "time", &varid));
00119  NC(nc_get_var_double(ncid, varid, times));
00120  NC(nc_inq_varid(ncid, "lat", &varid));
00121  NC(nc_get_var_double(ncid, varid, lats));
00122  NC(nc_inq_varid(ncid, "lon", &varid));
00123  NC(nc_get_var_double(ncid, varid, lons));
00124
00125  /* Get variable indices... */
00126  sprintf(varname, "%s_z", argv[4]);
00127  NC(nc_inq_varid(ncid, varname, &varid_z));
00128  sprintf(varname, "%s_p", argv[4]);
00129  NC(nc_inq_varid(ncid, varname, &varid_p));
00130  sprintf(varname, "%s_t", argv[4]);
00131  NC(nc_inq_varid(ncid, varname, &varid_t));
00132  sprintf(varname, "%s_q", argv[4]);
00133  h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00134
00135  /* Set dimensions... */
00136  count[0] = 1;
00137  count[1] = nlat;
00138  count[2] = nlon;
00139
00140  /* Create file... */
00141  printf("Write tropopause sample data: %s\n", argv[2]);
00142  if (!(out = fopen(argv[2], "w")))
00143      ERRMSG("Cannot create file!");
00144
00145  /* Write header... */
00146  fprintf(out,
00147          "# $1 = time [s]\n"
00148          "# $2 = altitude [km]\n"
00149          "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00150  for (iq = 0; iq < ctl.nq; iq++)
00151      fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00152              ctl.qnt_unit[iq]);
00153  fprintf(out, "# $5d = tropopause height [km]\n", 5 + ctl.nq);
00154  fprintf(out, "# $6d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00155  fprintf(out, "# $7d = tropopause temperature [K]\n", 7 + ctl.nq);
00156  fprintf(out, "# $8d = tropopause water vapor [ppv]\n\n", 8 + ctl.nq);
00157
00158  /* Loop over particles... */
00159  for (ip = 0; ip < atm->np; ip++) {
00160
00161      /* Check temporal ordering... */
00162      if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00163          ERRMSG("Time must be ascending!");
00164
00165      /* Check range... */
00166      if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00167          continue;
00168
00169      /* Read data... */
00170      it = locate_irr(times, (int) ntime, atm->time[ip]);
00171      if (it != it_old) {
00172          time0 = times[it];
00173          start[0] = (size_t) it;
00174          NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00175          for (ilon = 0; ilon < nlon; ilon++)
00176              for (ilat = 0; ilat < nlat; ilat++)
00177                  tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00178          NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00179          for (ilon = 0; ilon < nlon; ilon++)
00180              for (ilat = 0; ilat < nlat; ilat++)
00181                  tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00182          NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00183          for (ilon = 0; ilon < nlon; ilon++)
00184              for (ilat = 0; ilat < nlat; ilat++)
00185                  tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00186          if (h2o) {
00187              NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00188              for (ilon = 0; ilon < nlon; ilon++)
00189                  for (ilat = 0; ilat < nlat; ilat++)
00190                      tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00191          } else
00192              for (ilon = 0; ilon < nlon; ilon++)
00193                  for (ilat = 0; ilat < nlat; ilat++)
00194                      tropo_q0[ilon][ilat] = GSL_NAN;
00195
00196          time1 = times[it + 1];
00197          start[0] = (size_t) it + 1;
00198          NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00199          for (ilon = 0; ilon < nlon; ilon++)
00200              for (ilat = 0; ilat < nlat; ilat++)
00201                  tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00202          NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00203

```

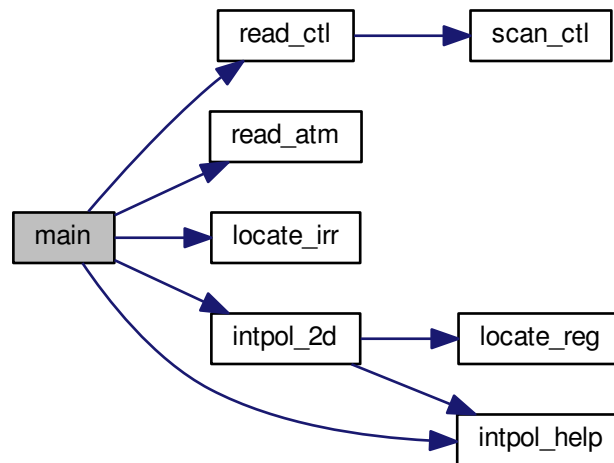


```

00204     for (ilon = 0; ilon < nlon; ilon++)
00205         for (ilat = 0; ilat < nlat; ilat++)
00206             tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00207     NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00208     for (ilon = 0; ilon < nlon; ilon++)
00209         for (ilat = 0; ilat < nlat; ilat++)
00210             tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00211     if (h2o) {
00212         NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00213         for (ilon = 0; ilon < nlon; ilon++)
00214             for (ilat = 0; ilat < nlat; ilat++)
00215                 tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00216     } else
00217         for (ilon = 0; ilon < nlon; ilon++)
00218             for (ilat = 0; ilat < nlat; ilat++)
00219                 tropo_q1[ilon][ilat] = GSL_NAN;;
00220 }
00221 it_old = it;
00222
00223 /* Interpolate... */
00224 z0 = intpol_2d(tropo_z0, lons, lats, nlon, nlat,
00225               atm->lon[ip], atm->lat[ip]);
00226 p0 = intpol_2d(tropo_p0, lons, lats, nlon, nlat,
00227               atm->lon[ip], atm->lat[ip]);
00228 t0 = intpol_2d(tropo_t0, lons, lats, nlon, nlat,
00229               atm->lon[ip], atm->lat[ip]);
00230 q0 = intpol_2d(tropo_q0, lons, lats, nlon, nlat,
00231               atm->lon[ip], atm->lat[ip]);
00232
00233 z1 = intpol_2d(tropo_z1, lons, lats, nlon, nlat,
00234               atm->lon[ip], atm->lat[ip]);
00235 p1 = intpol_2d(tropo_p1, lons, lats, nlon, nlat,
00236               atm->lon[ip], atm->lat[ip]);
00237 t1 = intpol_2d(tropo_t1, lons, lats, nlon, nlat,
00238               atm->lon[ip], atm->lat[ip]);
00239 q1 = intpol_2d(tropo_q1, lons, lats, nlon, nlat,
00240               atm->lon[ip], atm->lat[ip]);
00241
00242 z0 = intpol_help(time0, z0, time1, z1, atm->time[ip]);
00243 p0 = intpol_help(time0, p0, time1, p1, atm->time[ip]);
00244 t0 = intpol_help(time0, t0, time1, t1, atm->time[ip]);
00245 q0 = intpol_help(time0, q0, time1, q1, atm->time[ip]);
00246
00247 /* Write output... */
00248 fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00249       atm->lon[ip], atm->lat[ip]);
00250 for (iq = 0; iq < ctl.nq; iq++) {
00251     fprintf(out, " ");
00252     fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00253 }
00254 fprintf(out, " %g %g %g %g\n", z0, p0, t0, q0);
00255 }
00256
00257 /* Close files... */
00258 fclose(out);
00259 NC(nc_close(ncid));
00260
00261 /* Free... */
00262 free(atm);
00263
00264 return EXIT_SUCCESS;
00265 }

```

Here is the call graph for this function:



5.40 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) 2013-2019 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 /* Linear interpolation considering missing values. */
00033 double intpol_help(
00034     double x0,
00035     double y0,
00036     double x1,
00037     double y1,
00038     double x);
00039
00040 /* Bilinear horizontal interpolation. */
00041 double intpol_2d(
00042     float array[EX][EY],
00043     double lons[EX],
00044     double lats[EY],

```

```

00051     size_t nlon,
00052     size_t nlat,
00053     double lon,
00054     double lat);
00055
00056 /* -----
00057     Main...
00058     ----- */
00059
00060 int main(
00061     int argc,
00062     char *argv[]) {
00063
00064     ctl_t ctl;
00065
00066     atm_t *atm;
00067
00068     static FILE *out;
00069
00070     static char varname[LEN];
00071
00072     static double times[NT], lons[EX], lats[EY], time0, time1, z0, z1, p0, p1,
00073         t0, t1, q0, q1;
00074
00075     static float help[EX * EY], tropo_z0[EX][EY], tropo_z1[EX][EY],
00076         tropo_p0[EX][EY], tropo_p1[EX][EY], tropo_t0[EX][EY],
00077         tropo_t1[EX][EY], tropo_q0[EX][EY], tropo_q1[EX][EY];
00078
00079     static int ip, iq, it, it_old = -999, dimid[10], ncid,
00080         varid, varid_z, varid_p, varid_t, varid_q, h2o;
00081
00082     static size_t count[10], start[10], ntime, nlon, nlat, ilon, ilat;
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
00094     /* Read atmospheric data... */
00095     if (!read_atm(argv[5], &ctl, atm))
00096         ERRMSG("Cannot open file!");
00097
00098     /* Open tropopause file... */
00099     printf("Read tropopause data: %s\n", argv[3]);
00100     if (nc_open(argv[3], NC_NOWRITE, &ncid) != NC_NOERR)
00101         ERRMSG("Cannot open file!");
00102
00103     /* Get dimensions... */
00104     NC(nc_inq_dimid(ncid, "time", &dimid[0]));
00105     NC(nc_inq_dimlen(ncid, dimid[0], &ntime));
00106     if (ntime > NT)
00107         ERRMSG("Too many times!");
00108     NC(nc_inq_dimid(ncid, "lat", &dimid[1]));
00109     NC(nc_inq_dimlen(ncid, dimid[1], &nlat));
00110     if (nlat > EY)
00111         ERRMSG("Too many latitudes!");
00112     NC(nc_inq_dimid(ncid, "lon", &dimid[2]));
00113     NC(nc_inq_dimlen(ncid, dimid[2], &nlon));
00114     if (nlon > EX)
00115         ERRMSG("Too many longitudes!");
00116
00117     /* Read coordinates... */
00118     NC(nc_inq_varid(ncid, "time", &varid));
00119     NC(nc_get_var_double(ncid, varid, times));
00120     NC(nc_inq_varid(ncid, "lat", &varid));
00121     NC(nc_get_var_double(ncid, varid, lats));
00122     NC(nc_inq_varid(ncid, "lon", &varid));
00123     NC(nc_get_var_double(ncid, varid, lons));
00124
00125     /* Get variable indices... */
00126     sprintf(varname, "%s_z", argv[4]);
00127     NC(nc_inq_varid(ncid, varname, &varid_z));
00128     sprintf(varname, "%s_p", argv[4]);
00129     NC(nc_inq_varid(ncid, varname, &varid_p));
00130     sprintf(varname, "%s_t", argv[4]);
00131     NC(nc_inq_varid(ncid, varname, &varid_t));
00132     sprintf(varname, "%s_q", argv[4]);
00133     h2o = (nc_inq_varid(ncid, varname, &varid_q) == NC_NOERR);
00134
00135     /* Set dimensions... */
00136     count[0] = 1;
00137     count[1] = nlat;

```

```

00138     count[2] = nlon;
00139
00140     /* Create file... */
00141     printf("Write tropopause sample data: %s\n", argv[2]);
00142     if (!(out = fopen(argv[2], "w")))
00143         ERRMSG("Cannot create file!");
00144
00145     /* Write header... */
00146     fprintf(out,
00147             "# $1 = time [s]\n"
00148             "# $2 = altitude [km]\n"
00149             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00150     for (iq = 0; iq < ctl.nq; iq++)
00151         fprintf(out, "# $i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00152                 ctl.qnt_unit[iq]);
00153     fprintf(out, "# $d = tropopause height [km]\n", 5 + ctl.nq);
00154     fprintf(out, "# $d = tropopause pressure [hPa]\n", 6 + ctl.nq);
00155     fprintf(out, "# $d = tropopause temperature [K]\n", 7 + ctl.nq);
00156     fprintf(out, "# $d = tropopause water vapor [ppv]\n\n", 8 + ctl.nq);
00157
00158     /* Loop over particles... */
00159     for (ip = 0; ip < atm->np; ip++) {
00160
00161         /* Check temporal ordering... */
00162         if (ip > 0 && atm->time[ip] < atm->time[ip - 1])
00163             ERRMSG("Time must be ascending!");
00164
00165         /* Check range... */
00166         if (atm->time[ip] < times[0] || atm->time[ip] > times[ntime - 1])
00167             continue;
00168
00169         /* Read data... */
00170         it = locate_irr(times, (int) ntime, atm->time[ip]);
00171         if (it != it_old) {
00172
00173             time0 = times[it];
00174             start[0] = (size_t) it;
00175             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00176             for (ilon = 0; ilon < nlon; ilon++)
00177                 for (ilat = 0; ilat < nlat; ilat++)
00178                     tropo_z0[ilon][ilat] = help[ilat * nlon + ilon];
00179             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00180             for (ilon = 0; ilon < nlon; ilon++)
00181                 for (ilat = 0; ilat < nlat; ilat++)
00182                     tropo_p0[ilon][ilat] = help[ilat * nlon + ilon];
00183             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00184             for (ilon = 0; ilon < nlon; ilon++)
00185                 for (ilat = 0; ilat < nlat; ilat++)
00186                     tropo_t0[ilon][ilat] = help[ilat * nlon + ilon];
00187             if (h2o) {
00188                 NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00189                 for (ilon = 0; ilon < nlon; ilon++)
00190                     for (ilat = 0; ilat < nlat; ilat++)
00191                         tropo_q0[ilon][ilat] = help[ilat * nlon + ilon];
00192             } else
00193                 for (ilon = 0; ilon < nlon; ilon++)
00194                     for (ilat = 0; ilat < nlat; ilat++)
00195                         tropo_q0[ilon][ilat] = GSL_NAN;
00196
00197             time1 = times[it + 1];
00198             start[0] = (size_t) it + 1;
00199             NC(nc_get_vara_float(ncid, varid_z, start, count, help));
00200             for (ilon = 0; ilon < nlon; ilon++)
00201                 for (ilat = 0; ilat < nlat; ilat++)
00202                     tropo_z1[ilon][ilat] = help[ilat * nlon + ilon];
00203             NC(nc_get_vara_float(ncid, varid_p, start, count, help));
00204             for (ilon = 0; ilon < nlon; ilon++)
00205                 for (ilat = 0; ilat < nlat; ilat++)
00206                     tropo_p1[ilon][ilat] = help[ilat * nlon + ilon];
00207             NC(nc_get_vara_float(ncid, varid_t, start, count, help));
00208             for (ilon = 0; ilon < nlon; ilon++)
00209                 for (ilat = 0; ilat < nlat; ilat++)
00210                     tropo_t1[ilon][ilat] = help[ilat * nlon + ilon];
00211             if (h2o) {
00212                 NC(nc_get_vara_float(ncid, varid_q, start, count, help));
00213                 for (ilon = 0; ilon < nlon; ilon++)
00214                     for (ilat = 0; ilat < nlat; ilat++)
00215                         tropo_q1[ilon][ilat] = help[ilat * nlon + ilon];
00216             } else
00217                 for (ilon = 0; ilon < nlon; ilon++)
00218                     for (ilat = 0; ilat < nlat; ilat++)
00219                         tropo_q1[ilon][ilat] = GSL_NAN;
00220         }
00221         it_old = it;
00222
00223         /* Interpolate... */
00224         z0 = intpol_2d(tropo_z0, lons, lats, nlon, nlat,

```

```

00225         atm->lon[ip], atm->lat[ip]);
00226     p0 = intpol_2d(tropo_p0, lons, lats, nlon, nlat,
00227         atm->lon[ip], atm->lat[ip]);
00228     t0 = intpol_2d(tropo_t0, lons, lats, nlon, nlat,
00229         atm->lon[ip], atm->lat[ip]);
00230     q0 = intpol_2d(tropo_q0, lons, lats, nlon, nlat,
00231         atm->lon[ip], atm->lat[ip]);
00232
00233     z1 = intpol_2d(tropo_z1, lons, lats, nlon, nlat,
00234         atm->lon[ip], atm->lat[ip]);
00235     p1 = intpol_2d(tropo_p1, lons, lats, nlon, nlat,
00236         atm->lon[ip], atm->lat[ip]);
00237     t1 = intpol_2d(tropo_t1, lons, lats, nlon, nlat,
00238         atm->lon[ip], atm->lat[ip]);
00239     q1 = intpol_2d(tropo_q1, lons, lats, nlon, nlat,
00240         atm->lon[ip], atm->lat[ip]);
00241
00242     z0 = intpol_help(time0, z0, time1, z1, atm->time[ip]);
00243     p0 = intpol_help(time0, p0, time1, p1, atm->time[ip]);
00244     t0 = intpol_help(time0, t0, time1, t1, atm->time[ip]);
00245     q0 = intpol_help(time0, q0, time1, q1, atm->time[ip]);
00246
00247     /* Write output... */
00248     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
00249         atm->lon[ip], atm->lat[ip]);
00250     for (iq = 0; iq < ctl.nq; iq++) {
00251         fprintf(out, " ");
00252         fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00253     }
00254     fprintf(out, " %g %g %g %g\n", z0, p0, t0, q0);
00255 }
00256
00257 /* Close files... */
00258 fclose(out);
00259 NC(nc_close(ncid));
00260
00261 /* Free... */
00262 free(atm);
00263
00264 return EXIT_SUCCESS;
00265 }
00266
00267 /*****
00268 double intpol_help(
00269     double x0,
00270     double y0,
00271     double x1,
00272     double y1,
00273     double x) {
00274     double x) {
00275
00276     /* Linear interpolation... */
00277     if (gsl_finite(y0) && gsl_finite(y1))
00278         return LIN(x0, y0, x1, y1, x);
00279
00280     /* Nearest neighbour... */
00281     else {
00282         if (fabs(x - x0) < fabs(x - x1))
00283             return y0;
00284         else
00285             return y1;
00286     }
00287 }
00288
00289 /*****
00290 double intpol_2d(
00291     float array[EX][EY],
00292     double lons[EX],
00293     double lats[EY],
00294     size_t nlon,
00295     size_t nlat,
00296     double lon,
00297     double lat) {
00298     double lat) {
00299
00300     double aux0, aux1;
00301
00302     /* Adjust longitude... */
00303     if (lon < lons[0])
00304         lon += 360;
00305     else if (lon > lons[nlon - 1])
00306         lon -= 360;
00307
00308     /* Get indices... */
00309     int ix = locate_reg(lons, (int) nlon, lon);
00310     int iy = locate_reg(lats, (int) nlat, lat);
00311

```

```
00312  /* Interpolate in longitude... */
00313  aux0 = intpol_help(lons[ix], array[ix][iy],
00314                  lons[ix + 1], array[ix + 1][iy], lon);
00315  aux1 = intpol_help(lons[ix], array[ix][iy + 1],
00316                  lons[ix + 1], array[ix + 1][iy + 1], lon);
00317
00318  /* Interpolate in latitude... */
00319  return intpol_help(lats[iy], aux0, lats[iy + 1], aux1, lat);
00320 }
```

Index

- add_text_attribute
 - tropo.c, [281](#)
- atm_basename
 - ctl_t, [19](#)
- atm_conv.c, [28](#)
 - main, [29](#)
- atm_dist.c, [30](#)
 - main, [31](#)
- atm_dt_out
 - ctl_t, [19](#)
- atm_filter
 - ctl_t, [19](#)
- atm_gpfile
 - ctl_t, [19](#)
- atm_init.c, [39](#)
 - main, [39](#)
- atm_select.c, [43](#)
 - main, [43](#)
- atm_split.c, [47](#)
 - main, [47](#)
- atm_stat.c, [51](#)
 - main, [52](#)
- atm_stride
 - ctl_t, [19](#)
- atm_t, [3](#)
 - lat, [4](#)
 - lon, [4](#)
 - np, [4](#)
 - p, [4](#)
 - q, [4](#)
 - time, [4](#)
- atm_type
 - ctl_t, [19](#)
- balloon
 - ctl_t, [17](#)
- cache_t, [4](#)
 - iso_n, [6](#)
 - iso_ps, [6](#)
 - iso_ts, [6](#)
 - iso_var, [5](#)
 - tsig, [6](#)
 - up, [5](#)
 - usig, [6](#)
 - vp, [5](#)
 - vsig, [6](#)
 - wp, [5](#)
 - wsig, [6](#)
- cart2geo
 - libtrac.c, [64](#)
 - libtrac.h, [160](#)
- check_finite
 - libtrac.h, [160](#)
- cl
 - met_t, [27](#)
- clim_hno3
 - libtrac.c, [64](#)
 - libtrac.h, [161](#)
- clim_oh
 - libtrac.c, [65](#)
 - libtrac.h, [161](#)
- clim_tropo
 - libtrac.c, [66](#)
 - libtrac.h, [162](#)
- csi_basename
 - ctl_t, [19](#)
- csi_dt_out
 - ctl_t, [19](#)
- csi_lat0
 - ctl_t, [21](#)
- csi_lat1
 - ctl_t, [21](#)
- csi_lon0
 - ctl_t, [20](#)
- csi_lon1
 - ctl_t, [20](#)
- csi_modmin
 - ctl_t, [20](#)
- csi_nx
 - ctl_t, [20](#)
- csi_ny
 - ctl_t, [20](#)
- csi_nz
 - ctl_t, [20](#)
- csi_obsfile
 - ctl_t, [19](#)
- csi_obsmin
 - ctl_t, [20](#)
- csi_z0
 - ctl_t, [20](#)
- csi_z1
 - ctl_t, [20](#)
- ctl_t, [7](#)
 - atm_basename, [19](#)
 - atm_dt_out, [19](#)
 - atm_filter, [19](#)
 - atm_gpfile, [19](#)
 - atm_stride, [19](#)
 - atm_type, [19](#)
 - balloon, [17](#)
 - csi_basename, [19](#)
 - csi_dt_out, [19](#)
 - csi_lat0, [21](#)
 - csi_lat1, [21](#)
 - csi_lon0, [20](#)
 - csi_lon1, [20](#)
 - csi_modmin, [20](#)
 - csi_nx, [20](#)
 - csi_ny, [20](#)
 - csi_nz, [20](#)

csi_obsfile, 19
 csi_obsmin, 20
 csi_z0, 20
 csi_z1, 20
 direction, 15
 dt_met, 15
 dt_mod, 15
 ens_basename, 23
 grid_basename, 21
 grid_dt_out, 21
 grid_gpfile, 21
 grid_lat0, 22
 grid_lat1, 22
 grid_lon0, 22
 grid_lon1, 22
 grid_nx, 22
 grid_ny, 22
 grid_nz, 21
 grid_sparse, 21
 grid_z0, 21
 grid_z1, 21
 isosurf, 17
 met_dp, 16
 met_dt_out, 17
 met_dx, 15
 met_dy, 16
 met_geopot, 16
 met_np, 16
 met_p, 16
 met_sp, 16
 met_stage, 17
 met_sx, 16
 met_sy, 16
 met_tropo, 16
 molmass, 18
 nq, 11
 oh_chem, 18
 prof_basename, 22
 prof_lat0, 23
 prof_lat1, 23
 prof_lon0, 23
 prof_lon1, 23
 prof_nx, 23
 prof_ny, 23
 prof_nz, 22
 prof_obsfile, 22
 prof_z0, 23
 prof_z1, 23
 psc_h2o, 18
 psc_hno3, 18
 qnt_ens, 12
 qnt_format, 12
 qnt_h2o, 13
 qnt_hno3, 14
 qnt_iwc, 13
 qnt_lwc, 13
 qnt_m, 12
 qnt_name, 11
 qnt_o3, 13
 qnt_oh, 14
 qnt_p, 13
 qnt_pc, 14
 qnt_ps, 12
 qnt_pt, 12
 qnt_pv, 14
 qnt_r, 12
 qnt_rh, 14
 qnt_rho, 12
 qnt_stat, 15
 qnt_t, 13
 qnt_theta, 14
 qnt_tice, 14
 qnt_tnat, 15
 qnt_tsts, 15
 qnt_u, 13
 qnt_unit, 12
 qnt_v, 13
 qnt_vh, 14
 qnt_vz, 14
 qnt_w, 13
 qnt_z, 12
 species, 18
 stat_basename, 24
 stat_lat, 24
 stat_lon, 24
 stat_r, 24
 t_start, 15
 t_stop, 15
 tdec_strat, 18
 tdec_trop, 18
 turb_dx_strat, 17
 turb_dx_trop, 17
 turb_dz_strat, 17
 turb_dz_trop, 17
 turb_mesox, 17
 turb_mesoz, 18
 wet_depo, 18
 day2doy
 libtrac.c, 66
 libtrac.h, 163
 day2doy.c, 57
 main, 57
 direction
 ctl_t, 15
 doy2day
 libtrac.c, 67
 libtrac.h, 163
 doy2day.c, 59
 main, 59
 dt_met
 ctl_t, 15
 dt_mod
 ctl_t, 15
 ens_basename
 ctl_t, 23

geo2cart
 libtrac.c, 67
 libtrac.h, 164
get_met
 libtrac.c, 67
 libtrac.h, 164
get_met_help
 libtrac.c, 69
 libtrac.h, 165
get_met_replace
 libtrac.c, 70
 libtrac.h, 166
grid_basename
 ctl_t, 21
grid_dt_out
 ctl_t, 21
grid_gpfile
 ctl_t, 21
grid_lat0
 ctl_t, 22
grid_lat1
 ctl_t, 22
grid_lon0
 ctl_t, 22
grid_lon1
 ctl_t, 22
grid_nx
 ctl_t, 22
grid_ny
 ctl_t, 22
grid_nz
 ctl_t, 21
grid_sparse
 ctl_t, 21
grid_z0
 ctl_t, 21
grid_z1
 ctl_t, 21

h2o
 met_t, 28

intpol_2d
 tropo_sample.c, 290
intpol_help
 tropo_sample.c, 290
intpol_met_space_2d
 libtrac.c, 71
 libtrac.h, 167
intpol_met_space_3d
 libtrac.c, 70
 libtrac.h, 166
intpol_met_time_2d
 libtrac.c, 72
 libtrac.h, 169
intpol_met_time_3d
 libtrac.c, 72
 libtrac.h, 168
iso_n
 cache_t, 6
iso_ps
 cache_t, 6
iso_ts
 cache_t, 6
iso_var
 cache_t, 5
isosurf
 ctl_t, 17
iwc
 met_t, 28

jsec2time
 libtrac.c, 73
 libtrac.h, 169
jsec2time.c, 60
 main, 61

lat
 atm_t, 4
 met_t, 26
libtrac.c, 62
 cart2geo, 64
 clim_hno3, 64
 clim_oh, 65
 clim_tropo, 66
 day2doy, 66
 doy2day, 67
 geo2cart, 67
 get_met, 67
 get_met_help, 69
 get_met_replace, 70
 intpol_met_space_2d, 71
 intpol_met_space_3d, 70
 intpol_met_time_2d, 72
 intpol_met_time_3d, 72
 jsec2time, 73
 locate_irr, 73
 locate_reg, 74
 read_atm, 74
 read_ctl, 76
 read_met, 80
 read_met_cloud, 83
 read_met_extrapolate, 83
 read_met_geopot, 84
 read_met_help_2d, 86
 read_met_help_3d, 85
 read_met_ml2pl, 86
 read_met_periodic, 87
 read_met_pv, 88
 read_met_sample, 89
 read_met_surface, 90
 read_met_tropo, 91
 scan_ctl, 93
 spline, 94
 stddev, 95
 time2jsec, 95
 timer, 95
 write_atm, 96

- write_csi, 98
- write_ens, 100
- write_grid, 102
- write_prof, 104
- write_station, 106
- libtrac.h, 158
 - cart2geo, 160
 - check_finite, 160
 - clim_hno3, 161
 - clim_oh, 161
 - clim_tropo, 162
 - day2doy, 163
 - doy2day, 163
 - geo2cart, 164
 - get_met, 164
 - get_met_help, 165
 - get_met_replace, 166
 - intpol_met_space_2d, 167
 - intpol_met_space_3d, 166
 - intpol_met_time_2d, 169
 - intpol_met_time_3d, 168
 - jsec2time, 169
 - locate_irr, 170
 - locate_reg, 170
 - read_atm, 170
 - read_ctl, 173
 - read_met, 177
 - read_met_cloud, 179
 - read_met_extrapolate, 180
 - read_met_geopot, 180
 - read_met_help_2d, 182
 - read_met_help_3d, 182
 - read_met_ml2pl, 183
 - read_met_periodic, 184
 - read_met_pv, 184
 - read_met_sample, 185
 - read_met_surface, 187
 - read_met_tropo, 187
 - scan_ctl, 190
 - spline, 191
 - stddev, 191
 - time2jsec, 191
 - timer, 192
 - write_atm, 192
 - write_csi, 194
 - write_ens, 196
 - write_grid, 198
 - write_prof, 200
 - write_station, 203
- locate_irr
 - libtrac.c, 73
 - libtrac.h, 170
- locate_reg
 - libtrac.c, 74
 - libtrac.h, 170
- lon
 - atm_t, 4
 - met_t, 26
- lwc
 - met_t, 28
- main
 - atm_conv.c, 29
 - atm_dist.c, 31
 - atm_init.c, 39
 - atm_select.c, 43
 - atm_split.c, 47
 - atm_stat.c, 52
 - day2doy.c, 57
 - doy2day.c, 59
 - jsec2time.c, 61
 - met_map.c, 214
 - met_prof.c, 220
 - met_sample.c, 226
 - met_zm.c, 231
 - time2jsec.c, 238
 - trac.c, 256
 - tropo.c, 281
 - tropo_sample.c, 291
- met_dp
 - ctl_t, 16
- met_dt_out
 - ctl_t, 17
- met_dx
 - ctl_t, 15
- met_dy
 - ctl_t, 16
- met_geopot
 - ctl_t, 16
- met_map.c, 214
 - main, 214
- met_np
 - ctl_t, 16
- met_p
 - ctl_t, 16
- met_prof.c, 220
 - main, 220
- met_sample.c, 226
 - main, 226
- met_sp
 - ctl_t, 16
- met_stage
 - ctl_t, 17
- met_sx
 - ctl_t, 16
- met_sy
 - ctl_t, 16
- met_t, 24
 - cl, 27
 - h2o, 28
 - iwc, 28
 - lat, 26
 - lon, 26
 - lwc, 28
 - np, 26
 - nx, 26
 - ny, 26

- o3, [28](#)
- p, [26](#)
- pc, [27](#)
- pl, [28](#)
- ps, [26](#)
- pt, [27](#)
- pv, [28](#)
- t, [27](#)
- time, [26](#)
- u, [27](#)
- v, [27](#)
- w, [27](#)
- z, [27](#)
- zs, [27](#)
- met_tropo
 - ctl_t, [16](#)
- met_zm.c, [231](#)
 - main, [231](#)
- module_advection
 - trac.c, [240](#)
- module_decay
 - trac.c, [241](#)
- module_diffusion_init
 - trac.c, [242](#)
- module_diffusion_meso
 - trac.c, [242](#)
- module_diffusion_rng
 - trac.c, [244](#)
- module_diffusion_turb
 - trac.c, [245](#)
- module_isosurf
 - trac.c, [246](#)
- module_isosurf_init
 - trac.c, [246](#)
- module_meteo
 - trac.c, [247](#)
- module_oh_chem
 - trac.c, [252](#)
- module_position
 - trac.c, [250](#)
- module_sedi
 - trac.c, [251](#)
- module_wet_deposition
 - trac.c, [253](#)
- molmass
 - ctl_t, [18](#)
- np
 - atm_t, [4](#)
 - met_t, [26](#)
- nq
 - ctl_t, [11](#)
- nvtxmc.h, [237](#)
- nx
 - met_t, [26](#)
- ny
 - met_t, [26](#)
- o3
 - met_t, [28](#)
- oh_chem
 - ctl_t, [18](#)
- p
 - atm_t, [4](#)
 - met_t, [26](#)
- pc
 - met_t, [27](#)
- pl
 - met_t, [28](#)
- prof_basename
 - ctl_t, [22](#)
- prof_lat0
 - ctl_t, [23](#)
- prof_lat1
 - ctl_t, [23](#)
- prof_lon0
 - ctl_t, [23](#)
- prof_lon1
 - ctl_t, [23](#)
- prof_nx
 - ctl_t, [23](#)
- prof_ny
 - ctl_t, [23](#)
- prof_nz
 - ctl_t, [22](#)
- prof_obsfile
 - ctl_t, [22](#)
- prof_z0
 - ctl_t, [23](#)
- prof_z1
 - ctl_t, [23](#)
- ps
 - met_t, [26](#)
- psc_h2o
 - ctl_t, [18](#)
- psc_hno3
 - ctl_t, [18](#)
- pt
 - met_t, [27](#)
- pv
 - met_t, [28](#)
- q
 - atm_t, [4](#)
- qnt_ens
 - ctl_t, [12](#)
- qnt_format
 - ctl_t, [12](#)
- qnt_h2o
 - ctl_t, [13](#)
- qnt_hno3
 - ctl_t, [14](#)
- qnt_iwc
 - ctl_t, [13](#)
- qnt_lwc
 - ctl_t, [13](#)
- qnt_m

- ctl_t, 12
- qnt_name
 - ctl_t, 11
- qnt_o3
 - ctl_t, 13
- qnt_oh
 - ctl_t, 14
- qnt_p
 - ctl_t, 13
- qnt_pc
 - ctl_t, 14
- qnt_ps
 - ctl_t, 12
- qnt_pt
 - ctl_t, 12
- qnt_pv
 - ctl_t, 14
- qnt_r
 - ctl_t, 12
- qnt_rh
 - ctl_t, 14
- qnt_rho
 - ctl_t, 12
- qnt_stat
 - ctl_t, 15
- qnt_t
 - ctl_t, 13
- qnt_theta
 - ctl_t, 14
- qnt_tice
 - ctl_t, 14
- qnt_tnat
 - ctl_t, 15
- qnt_tsts
 - ctl_t, 15
- qnt_u
 - ctl_t, 13
- qnt_unit
 - ctl_t, 12
- qnt_v
 - ctl_t, 13
- qnt_vh
 - ctl_t, 14
- qnt_vz
 - ctl_t, 14
- qnt_w
 - ctl_t, 13
- qnt_z
 - ctl_t, 12
- read_atm
 - libtrac.c, 74
 - libtrac.h, 170
- read_ctl
 - libtrac.c, 76
 - libtrac.h, 173
- read_met
 - libtrac.c, 80
 - libtrac.h, 177
- read_met_cloud
 - libtrac.c, 83
 - libtrac.h, 179
- read_met_extrapolate
 - libtrac.c, 83
 - libtrac.h, 180
- read_met_geopot
 - libtrac.c, 84
 - libtrac.h, 180
- read_met_help_2d
 - libtrac.c, 86
 - libtrac.h, 182
- read_met_help_3d
 - libtrac.c, 85
 - libtrac.h, 182
- read_met_ml2pl
 - libtrac.c, 86
 - libtrac.h, 183
- read_met_periodic
 - libtrac.c, 87
 - libtrac.h, 184
- read_met_pv
 - libtrac.c, 88
 - libtrac.h, 184
- read_met_sample
 - libtrac.c, 89
 - libtrac.h, 185
- read_met_surface
 - libtrac.c, 90
 - libtrac.h, 187
- read_met_tropo
 - libtrac.c, 91
 - libtrac.h, 187
- rng
 - trac.c, 262
- scan_ctl
 - libtrac.c, 93
 - libtrac.h, 190
- species
 - ctl_t, 18
- spline
 - libtrac.c, 94
 - libtrac.h, 191
- stat_basename
 - ctl_t, 24
- stat_lat
 - ctl_t, 24
- stat_lon
 - ctl_t, 24
- stat_r
 - ctl_t, 24
- stddev
 - libtrac.c, 95
 - libtrac.h, 191
- t
 - met_t, 27
- t_start

- ctl_t, 15
- t_stop
 - ctl_t, 15
- tdec_strat
 - ctl_t, 18
- tdec_trop
 - ctl_t, 18
- time
 - atm_t, 4
 - met_t, 26
- time2jsec
 - libtrac.c, 95
 - libtrac.h, 191
- time2jsec.c, 238
 - main, 238
- timer
 - libtrac.c, 95
 - libtrac.h, 192
- trac.c, 239
 - main, 256
 - module_advection, 240
 - module_decay, 241
 - module_diffusion_init, 242
 - module_diffusion_meso, 242
 - module_diffusion_rng, 244
 - module_diffusion_turb, 245
 - module_isosurf, 246
 - module_isosurf_init, 246
 - module_meteo, 247
 - module_oh_chem, 252
 - module_position, 250
 - module_sedi, 251
 - module_wet_deposition, 253
 - rng, 262
 - write_output, 254
- tropo.c, 281
 - add_text_attribute, 281
 - main, 281
- tropo_sample.c, 290
 - intpol_2d, 290
 - intpol_help, 290
 - main, 291
- tsig
 - cache_t, 6
- turb_dx_strat
 - ctl_t, 17
- turb_dx_trop
 - ctl_t, 17
- turb_dz_strat
 - ctl_t, 17
- turb_dz_trop
 - ctl_t, 17
- turb_mesox
 - ctl_t, 17
- turb_mesoz
 - ctl_t, 18
- u
 - met_t, 27
- up
 - cache_t, 5
- usig
 - cache_t, 6
- v
 - met_t, 27
- vp
 - cache_t, 5
- vsig
 - cache_t, 6
- w
 - met_t, 27
- wet_depo
 - ctl_t, 18
- wp
 - cache_t, 5
- write_atm
 - libtrac.c, 96
 - libtrac.h, 192
- write_csi
 - libtrac.c, 98
 - libtrac.h, 194
- write_ens
 - libtrac.c, 100
 - libtrac.h, 196
- write_grid
 - libtrac.c, 102
 - libtrac.h, 198
- write_output
 - trac.c, 254
- write_prof
 - libtrac.c, 104
 - libtrac.h, 200
- write_station
 - libtrac.c, 106
 - libtrac.h, 203
- wsig
 - cache_t, 6
- z
 - met_t, 27
- zs
 - met_t, 27